

AD-A239 358



REPORT DOCUMENTATION PAGE			FORM 288-104 MAY 1962 EDITION GSA GEN. REG. NO. 27	
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE	3. REPORT TYPE AND DATES COVERED THESIS/ DISSERTATION	
4. TITLE AND SUBTITLE Interactive Constraint-Based Solid Modeling as a Design Tool			5. FUNDING NUMBERS	
6. AUTHOR(S) Kenneth L. Toblin, Captain				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFIT Student Attending: Ohio State University			8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/CI/CIA-91-046	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFIT/CI Wright-Patterson AFB OH 45433-6583			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release IAW 190-1 Distributed Unlimited ERNEST A. HAYGOOD, 1st Lt, USAF Executive Officer			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words)				
14. SUBJECT TERMS			15. NUMBER OF PAGES 148	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT	

THESIS ABSTRACT

AUTHOR: Kenneth Lee Tobin

MILITARY RANK AND BRANCH: Captain, USAF

TITLE OF THESIS: Interactive Constraint-Based Solid Modeling as a Design Tool.

DATE: 1991

NUMBER OF PAGES: 148

DEGREE AWARDED: Master of Architecture (M.ARCH)

NAME OF INSTITUTION: The Ohio State University.

ABSTRACT:

Architectural design knowledge which may take the form of constraints and how it can be incorporated into the solid modeling process is explored and discussed. This theoretical exploration leads to an illustration of a functional general purpose three-dimensional solid modeler which utilizes design knowledge as constraints upon the interactive solid modeling process. From this illustration, it is shown that the incorporation of constraints into Computer-Aided Architectural Design can effectively assist in the early stages of the design process. A prototypical application is presented which provides an example in which the definition and implementation of specific design knowledge constrains or characterizes the generative and interactive behavior of user definable three-dimensional entities. The basis of the thesis lies in the ability to create a design space, or building envelope, and to allow the interactive modeling of conceptual elements within, and constrained to, that design space, including the realistic interaction between the entities themselves.

A prototypical application, called C•Mod, has been implemented as an extension to an existing educational modeling shell. C•Mod operates on the Macintosh II® micro-computer and can effectively illustrate constraint-based interactive modeling as a conceptual design tool.

KEY PRIMARY AND SECONDARY SOURCES:

Coyne, R.D., M.A.Rosenman, A.D.Radford, M.Balachandran, and J.S.Gero. (1990). *Knowledge-Based Design Systems*, Addison-Wesley, Reading, Massachusetts.

Foley, J. D., A. van Dam, S.K.Fiener, J.F.Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company, Reading Massachusetts.

Mitchell, W.J. (1990). *The Logic of Architecture*, The MIT Press, Cambridge, Massachusetts.

Yessios, C.I. (1987). "Architectural modeling and knowledge systems." In *Proceedings, NCGA Computer Graphics*.

91 8 07 138

91-07260



**INTERACTIVE CONSTRAINT-BASED SOLID MODELING
AS A DESIGN TOOL**

A Thesis

Presented in Partial Fulfillment of the Requirements for
the degree Master of Architecture in the
Graduate School of The Ohio State University

by

Kenneth Lee Tobin, B.ARCH. B.S.

* * * * *

The Ohio State University

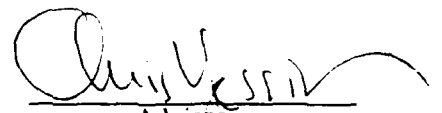
1991

Master's Examination Committee:

Christos I. Yessios

Benjamin Gianni

Approved by


Advisor
Department of Architecture



Author	Tobin, Kenneth Lee
Title	Interactive Constraint-Based Solid Modeling As a Design Tool
Department	Architecture
College	Graduate School
Year	1991
Thesis	Yes
Dist	Yes
A-1	

Copyright by
Kenneth Lee Tobin
1991

To my family

ACKNOWLEDGMENTS

I wish to thank the Department of Defense, United States Air Force, and the Air Force Institute of Technology for their support, and for the opportunity I have received to pursue an advanced education. I would like to extend my appreciation to The Ohio State University, Department of Architecture, Computer Aided Architectural Design program, for the use of their MacMod844 educational shell. I wish to acknowledge, and extend a deep appreciation to, Dr. Chris I. Yessios for his contributions and support throughout my graduate studies, and for his guidance and advice during this thesis project. Thanks to Benjamin Gianni for his continuous enthusiasm and support while serving as a member of my thesis committee. I would also like to thank my colleagues in the CAAD program for their continued support and friendship. Most importantly, I would like to express my love and appreciation to my wife, Susan, for her understanding and support.

TABLE OF CONTENTS

ACKNOWLEDGMENTS.....	iii
VITA.....	iv
LIST OF FIGURES	ix
PREFACE.....	xvi
CHAPTER	PAGE
I INTRODUCTION	1
1.1 Overview	2
1.2 Application to Current CAAD Theory	4
1.3 Historical Basis.....	7
1.4 Goals and Expectations.....	8
II CONSTRAINT-BASED MODELING OPERATIONS	11
2.1 Knowledge Specification.....	12
2.2 Knowledge Manipulation.....	12
2.3 Constraint Modeling	14
2.3.1 Entity Characteristics	14
2.3.1.1 Spatial Representation.....	15
2.3.1.2 Dimensional Attributes.....	16
2.3.1.3 Spatial Characteristics.....	17
2.3.1.4 Rotational Attributes.....	18
2.3.1.5 Mobility Characteristic	19
2.3.2 Relational Characteristics.....	21
2.3.2.1 Proximity Relationships	21
2.3.2.2 Containment Relationships	22
2.3.2.3 Associative Relationships.....	23
2.3.3 Identification Characteristics.....	26

	2.3.3.1	Type Definition	26
2.4		Knowledge Query	27
2.5		Direction.....	27
III		C•MOD - SYSTEM OVERVIEW AND USER'S MANUAL	28
3.1		Introduction to C•Mod.....	29
	3.1.1	Activating the program	29
	3.1.2	Graphical Environment.....	30
3.2		Overview of the Menu and Command Structure.....	31
	3.2.1	The Apple Menu.....	31
	3.2.2	The File Menu	32
	3.2.3	The Edit Menu.....	33
	3.2.4	The Height Menu.....	34
	3.2.5	The Views Menu	35
	3.2.6	The Modes Menu.....	36
	3.2.7	The Options Menu	37
3.3		Discussion of Basic Capabilities.....	38
	3.3.1	Two-Dimensional Drawing	39
	3.3.2	Three-Dimensional Extrusions	40
	3.3.3	Three-Dimensional Convergence.....	41
	3.3.4	Three-Dimensional Objects of Revolution	42
	3.3.5	Geometric Editing.....	43
	3.3.6	Topological Editing.....	43
	3.3.7	Textures	44
	3.3.8	Void Modeling.....	45
	3.3.9	Color - Surface Shading.....	46
	3.3.10	Constraint-Based Modeling.....	47
3.4		C•Mod Constraint Operations.....	48
	3.4.1	Set and View-Modify.....	49
	3.4.1.1	Entity Characteristics.....	50
	3.4.1.1.1	Spatial Representation.....	50
	3.4.1.1.2	Dimensional Characteristics	51
	3.4.1.1.3	Spatial Characteristics	53
	3.4.1.1.4	Rotational Attributes.....	53
	3.4.1.1.5	Mobility Characteristics	54

	3.4.1.2	Relational Characteristics	56
	3.4.1.2.1	Proximity Relationships.....	56
	3.4.1.2.2	Containment Relationships	57
	3.4.1.2.3	Associative Relationship	59
	3.4.1.3	Identification Attribute	64
	3.4.1.3.1	Type Definition.....	64
	3.4.2	Store Type and Retrieve Type	65
	3.4.2.1	Store Type.....	65
	3.4.2.2	Retrieve Type	66
	3.4.3	Query	66
IV		INTERNAL WORKINGS AND ALGORITHMS.....	68
4.1		Discussion of the Internal Composition	68
4.1.1		Object Data Structure	69
4.1.2		Constraint Data Structure.....	73
4.2		Algorithmic Outline of Required Procedures.....	82
4.2.1		Knowledge Specification.....	82
	4.2.1.1	User Interface	82
	4.2.1.2	Dialog Handler.....	84
4.2.2		Entity Generation.....	86
	4.2.2.1	Generation within Dimensional Constraints.....	86
4.2.3		Constraint Modeling	92
	4.2.3.1	Entity Manipulation With Constraints.....	92
	4.2.3.2	Constraint Satisfaction	93
	4.2.3.3	Detection and Interference Determination.....	96
V		C•MOD APPLICATIONS	101
5.1		Conceptual Design of an Architectural Element.....	101
5.2		Conceptual Design of Building Components	105
5.3		Conceptual Design Utilizing the Design Space	112
5.4		Summary of Applications	117
VI		EXTENSIONS AND FUTURE DIRECTIONS	118
6.1		Extensions to C•Mod	118
	6.1.1	C•Mod Operations	119

6.1.2	Algorithm Coherence	119
6.2	Extensions to Constraint-Based Modeling Operations	120
6.2.1	Entity Characteristic Operations	120
6.2.2	Relational Characteristic Operations	121
6.2.3	Physical Laws of Nature	122
6.3	Future Direction of Constraint-Based Solid Modeling	123
VII	CONCLUSIONS.....	124
	LIST OF REFERENCES.....	127

LIST OF FIGURES

FIGURES	PAGE
1. Preliminary site information	2
2. Supplementary site information	3
3. Design space constrained by site limitations	3
4. Illustration of modeling within the design space.	4
5. Components of the modeling process	5
6. Flow diagram of the simulation process of achieving problem satisfaction	6
7. Flow diagram of the optimization process of achieving problem satisfaction	6
8. Flow diagram of the interactive constraint process of achieving problem satisfaction	7
9. Design knowledge of a space	12
10. Design knowledge of a solid	12
11. Constraints upon a single entity and stored as a brick definition	13
12. Preset constraints applied to all subsequently generated entities	13
13. Representation of a solid entity and a space entity	15
14. Three-dimensional representation of solids and spaces	15
15. Visualization of the width, length, and height dimensional criteria	16
16. Three-dimensional representation of the bounding box and dimensional data ..	16
17. Bounding box and solid entity in three-space	17
18. Replication of a solid entity in the width dimension	17
19. An illustration of the area characteristic	18

20.	An illustration of the volume characteristic	18
21.	Visualization of the limits on the x-y reference plane	19
22.	Example of an object rotated about the origin, constrained to limits	19
23.	Illustration of an interaction between a translating entity and an entity with fixed mobility	20
24.	Illustration of an interaction between a translating entity and an entity with free mobility	21
25.	An example of the proximity zone and it's relationship with an entity	22
26.	Example of the containment relationship and it's interaction between entities ..	23
27.	Example of the attract association and secondary entity with free mobility	24
28.	Example of the attract association and secondary entity with fixed mobility	24
29.	Example of the repel association and secondary entity with free mobility	25
30.	Example of the repel association and secondary entity with fixed mobility	25
31.	An illustration of the type definitions	26
32.	C•Mod prototypical application	28
33.	C•Mod icons	29
34.	C•Mod graphic environment	30
35.	The C•Mod menu bar	31
36.	Apple menu	31
37.	About Constraint Mod dialog box	31
38.	File menu	32
39.	File icon types	32
40.	Save file dialog box	33
41.	Load file dialog box	33
42.	Edit menu	33
43.	An illustration of the reference planes viewed from 45-45 angle	34
44.	The height menu	34

45.	The views menu	35
46.	The effects of the view command on the reference plane	35
47.	The modes menu	36
48.	The options menu	38
49.	Basic user interface in the graphical environment	39
50.	The tool box of icons in the two-dimensional drawing mode	40
51.	An example of drawing in the two-dimensional drawing mode	40
52.	The tool box of icons in the extrude mode	41
53.	An example of creating solid objects in the extrude mode	41
54.	The tool box of icons in the converge mode	41
55.	An example of creating solid objects in the converge mode	41
56.	The tool box of icons in the revolve mode	42
57.	An example of creating solid objects in the revolve mode	42
58.	The tool box of icons in the g-edit mode	43
59.	An example of editing solid objects in the g-edit mode	43
60.	The tool box of icons in the t-edit mode	44
61.	An example of editing solid objects in the t-edit mode	44
62.	The tool box of icons in the textures mode	45
63.	An example of rendering solid objects in the texture mode	45
64.	The tool box of icons in the void model mode	45
65.	An example of generating a two-dimensional void model and a solid roof element from the void	45
66.	The tool box of icons in the colors mode	46
67.	An example of rendering solid objects in the colors mode	46
68.	The tool box of icons in the constraints mode	47
69.	An example of editing solid objects (g-edit) with apply constraints active	47

70.	The constraint mode sub-menu	48
71.	Dialog box for the spatial representation constraint	50
72.	Illustration of the solid and space representation	50
73.	Dialog box for the dimensional characteristics constraint	51
74.	Illustration of the construction lines provided to visualize the constraints	51
75.	An illustration of the interactive generation of an entity within a dimensional set of constraints	52
76.	An illustration of an object created within the dimensional constraints	52
77.	Geometric editing within the limitations of the dimensional constraints	52
78.	Visualization of the bounding box method of maintaining dimensional data ...	52
79.	Dialog box for the spatial characteristics constraint	53
80.	Illustration of the application of spatial characteristics to the object	53
81.	Dialog box for the rotational attributes constraint	54
82.	Illustration of the application of rotational attributes to the object	54
83.	Dialog box for the mobility characteristics constraint	55
84.	Assignments of the mobility characteristic of two solid entities	55
85.	An illustration of the dynamic interaction between an object translating in three-space which confronts a fixed object	55
86.	An illustration of the dynamic interaction between an object translating in three-space which confronts a free object	55
87.	Dialog box for the proximity relationship for solid entities	57
88.	The interaction between entities with and without proximity relationships	57
89.	Dialog box for the proximity relationship for space entities	57
90.	The interaction between a space entity and two solid entities	57
91.	Dialog box for the containment relationship for solid entities	58
92.	Dialog box for the containment relationship for space entities	58
93.	An illustration of the containment constraint on the modeling process	58

94. Dialog box for the associative relationship between entities	59
95. The offset from the proximity zone which determines activation of the associative relationship	59
96. The anticipated translation and resultant interaction involving an entity with free mobility and an attract association	60
97. The resulting interaction between an entity with free mobility and an attract association	60
98. The anticipated translation and resultant interaction involving an entity with fixed mobility and an attract association	61
99. The resulting interaction between an entity with fixed mobility and an attract association	61
100. The anticipated translation and resultant interaction involving an entity with free mobility and a repel association	62
101. The resulting interaction between an entity with free mobility and a repel association	62
102. The anticipated translation and resultant interaction involving an entity with fixed mobility and a repel association	63
103. The resulting interaction between an entity with fixed mobility and a repel association	63
104. The type definition dialog box	64
105. The type definition file	65
106. Store file dialog box	66
107. Retrieve file dialog box	66
108. The query dialog box	67
109. Topological levels of an object	69
110. Diagram of the object data structure	70
111. Diagram of the constraint data structure	74
112. An example of the internal constraint database for an entity	81
113. Illustration of the interface calls to properly invoke the dialog handler	83
114. Illustration of the dialog handler	85

115. Proximity/Association resultant matrix	97
116. Generation of an entity within dimensional constraints	102
117. Results illustrating the generated entity	102
118. Topological editing of the entity	102
119. Geometric edition of the inserted segments	102
120. Subsequent topological and geometric editing to create an architectural element	103
121. Further geometric editing of a face of the architectural entity	103
122. Subsequent geometric editing within dimensional constraints	104
123. Rotation of the element on the x-y plane	104
124. Illustration of the data representation after rotation	104
125. Illustration of completed architectural element modeled in C•Mod	104
126. A rendered illustration of an architectural element, which was created utilizing the constraint operations of C•Mod	105
127. An illustration of three rooms created with the void modeling operations	106
128. Geometric editing of room A, and subsequent translation of B and C	106
129. Final placement of three rooms constrained by the solidity of the entities	107
130. Generation and placement of a horizontal solid entity	107
131. Geometric editing of the roof entity and resulting interaction with the rooms ..	108
132. An illustration of element generation within dimensional constraints with the replication operations activated	108
133. The results of multiple column generation by replication	109
134. Topological and geometric editing within dimensional constraints	109
135. Continued topological and geometric editing of the roof element	110
136. Translation of the columns constrained by the mobility of the roof	110
137. Elevation view of the placement of a base for the columns restricted by the mobility of the roof	110

138. Results of the constraint-based interactive modeling utilized for a conceptual design	110
139. Rendered results of a schematic design solution utilizing C•Mod	111
140. Creation of a design space within dimensional constraints	112
141. Modeling the design space interactively by inserting a segment in the face of a space	112
142. Establishing the shadow/sunlight restriction upon the design space	113
143. The generation of a solid entity within the design space	113
144. Translation of the solid entity within the design space, and restricted to the constraints of the design space	113
145. Interactive modeling of the solid entity within and constrained to the design space	113
146. Subsequent geometric editing of the solid entity constrained to the height and shadow setback of the design space	114
147. Creation of a second entity within the design space	114
148. Vertical translation of the solid entity constrained to the height limitations of the design space	115
149. Association attribute of attraction established between the two solid entities ..	115
150. Generation of additional solid entities within the design space	115
151. Completion of the conceptual design within the design space	115
152. An illustration of the rendered results of the conceptual design created by the prototypical application C•Mod	116

PREFACE

The prototypical constraint-based solid modeler, C•Mod, presented in this thesis, is an extension to an existing educational solid modeler, MacMod844. MacMod844 is a solid and void modeling program used by the Department of Architecture, Computer-Aided Architectural Design Department, as the fundamental method of studying and implementing concepts of two-dimensional drawing, and three-dimensional solid visualization and representation, as well as the application of these concepts to architectural design. This program was written in part by the author, and by other graduate students in the CAAD program, as a series of educational exercises. The results of the MacMod844 solid modeler stem from research provided by Dr. Christos Yessios, and by the completion of the exercises accomplished during a six quarter sequence of architecture classes provided by the Department of Architecture.

CHAPTER I

INTRODUCTION

In 1972, Nicholas Negroponte wrote "Computer-aided design cannot occur without machine intelligence - and would be dangerous without it" (Negroponte,1972). This early insight to the possible introduction of knowledge-based design systems to architectural design has led the way for many academic theories and applications. However, until recently, the profession has greatly ignored the potential of such systems, relying on the utilization of computer technology to aid in the production phases of the design process - mainly drafting. More recently, the advances in computer technology, predominantly in the micro-computer arena, has led to a greater understanding, representation, and manipulation of knowledge in a manner which has made it possible to construct knowledge-based design systems capable of aiding the designer in the early phases of the design process. According to R.D.Coyne, "The emphasis now is in finding methods of appropriating and rendering operable the knowledge available to designers" (Coyne,1990).

Knowledge, for the purpose of this discussion, includes the process, declaration, and state of having information pertaining to a particular entity or group of entities. It becomes the representation of the ability to gain, apply, and state the behavioral attributes an object or entity may possess. The interaction between an entity and the world, or between entities, is therefore limited or constrained to the knowledge representing that entity. A constraint, therefore, becomes the ability to limit the use, generation or manipulation of entities within the established bounds of knowledge.

This thesis explores the use of design knowledge with the intent of establishing a more logical and efficient method of utilizing computers to assist in the design activity. A prototypical application, C*Mod, is presented to support the fundamental applications of constraint-based solid modeling, and to provide a visual representation of the concepts presented. One such concept is the use of design knowledge to interactively constrain the design space, the space in which design is allowed to emerge. By implementing this ability to provide and utilize knowledge, it is possible to develop a constraint-based solid modeler for architectural applications which provides feedback to the designer in "real time".

1.1 Overview

The goal of this application is to provide an avenue in which knowledge can be obtained, synthesized, and utilized to constrain the interactive generation, manipulation and behavior of geometric solid elements. By utilizing available design knowledge to constrain geometric entities, it will be possible to bring a basic three-dimensional solid modeler closer to the early stages of the design process, and allow a CAAD system to be more of an aid to the designer.

Constraining the interactive design process can be illustrated by considering a conceptual design of a building on a site which zoning regulations impose limitations to the design. Initially, there are limitations to the site itself, boundaries which limit the breadth of

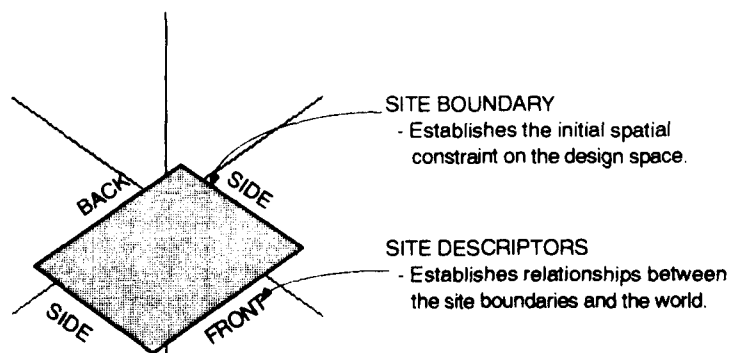


Figure 1. Preliminary site information

construction and descriptive symbols which establish relationships and adjacencies. Typically, there may be setback requirements for the front, side, and back yards, establishing a minimum distance which a building may be placed with respect to the site boundaries. Figure 1, above, illustrates this preliminary site information. There may be incremental requirements such as

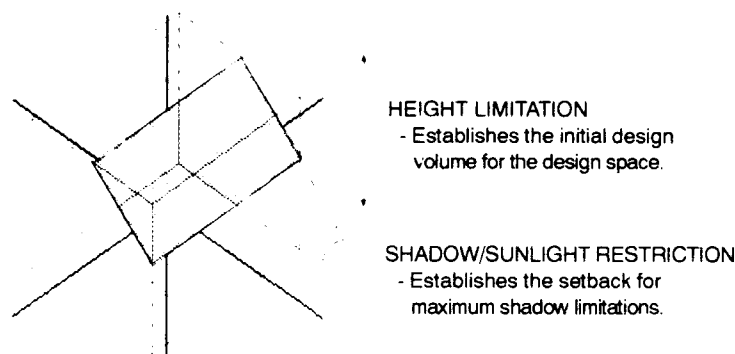


Figure 2. Supplemental site information

height limitations and sunlight restrictions. Figure 2, above, provides a graphic representation of these secondary site restrictions. Additionally, legal constraints, such as right-of-ways, may be imposed, further restricting the site. By applying these limitations upon a geometrical entity, we have created a design space which contains knowledge about the allowable design volume. An illustration of this design space with the imposed site restrictions is found in figure 3.

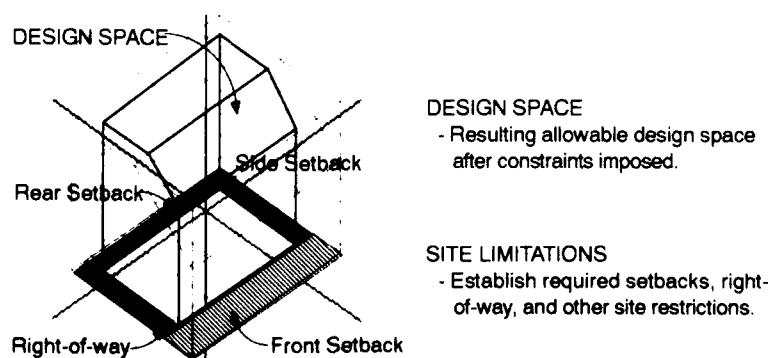


Figure 3. Design space constrained by site limitations

Knowledge of the design space can then be imposed on the interactive three-dimensional solid modeling process, allowing only acceptable solutions to be generated and manipulated within this allowable design space. Figure 4, below, graphically illustrates the results of such interactive modeling within the constraints of site design knowledge.

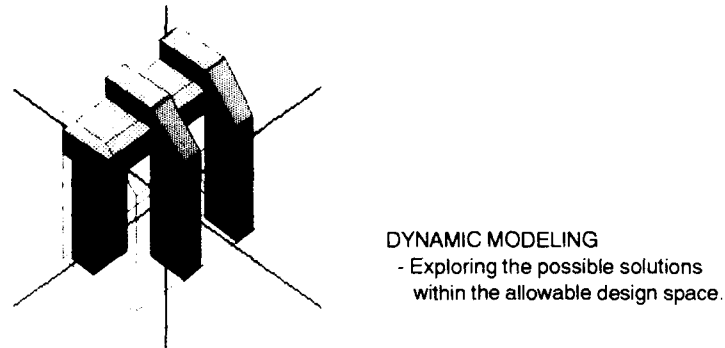


Figure 4. Illustration of modeling within the design space.

The fundamental basis for this thesis results in this ability to constrain, with the use of design knowledge, the generation, manipulation, and behavior of geometrical entities. By allowing the user to interactively identify and specify design knowledge, the interaction of these geometrical entities reduces to constraint satisfaction in which the lower level decisions are required to be met. Therefore, the benefit of such constraint-based modeling is that the designer has imposed these lower level decisions at the conceptual stage of the design process, and provides him with the opportunity to utilize the computer and design knowledge, early in this process.

1.2 Application to Current CAAD Theory

One of the current goals of CAAD is to enhance the ability to design by aiding in the design process. The architectural design process involves the analysis, synthesis, refinement, and implementation of a set of requirements, into a viable design solution which satisfies those goals (AIA,1987). Current CAAD theory involves the modeling of, in terms of a computer

model, this process of design. Many academic theorists and practitioners have proposed methods of modeling this process. Among them, two basic methods of modeling the design process have emerged; simulation, which is a problem solving approach to design in which a design solution is postulated and then evaluated against the design goals for compliance, and optimization, where the performance criteria, constraints, and decisions are stated, and the best possible design is produced. Both methods are briefly presented in order to juxtapose the method which this paper is based.



Figure 5. Components of the modeling process.

Both methods incorporate four major elements into the process; design goals, which state the specific goals which the design is required to obtain, design objectives, which state the specific elements the design must meet in order to satisfy the design goals, performance variables, which state the specific variables which must acquire some values, or certain ranges of values, which will satisfy the objectives, and decision variables which state the specific assignment of values to the performance variables (Coyne, 1990). It is the use of these elements, and how they are implemented, which constitutes the current theory of knowledge-based design systems applicable to CAAD.

The simulation process involves the postulation of a solution to a design problem in which the decision variables are made. The performance variables are determined from the decision variables, and evaluated against the design objectives to determine whether the solution meets the design goals. If it does, then the solution satisfies the problem. If it does not meet the design goals, the solution must return to the decision variable state, and the decision variables must be modified. This model of the process proposes a solution to a given problem, then evaluates it against the design goals. (Coyne, 1990; Mitchell, 1990)

SIMULATION PROCESS: (design postulated then evaluated)

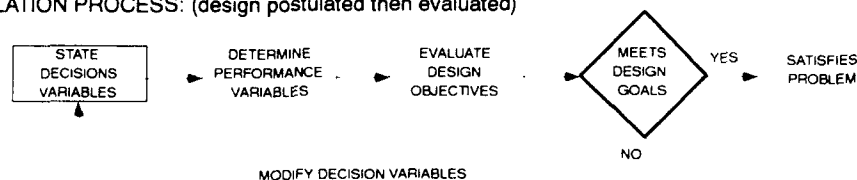


Figure 6. Flow diagram of the simulation process of achieving problem satisfaction.

The optimization process involves the generation of a design solution from a search through all of the possible states which meet the stated decision variables and performance variables. This model of the process evaluates the design goals and variables, and provides a theoretically optimal solution which satisfies the problem. (Coyne, 1990; Mitchell, 1990)

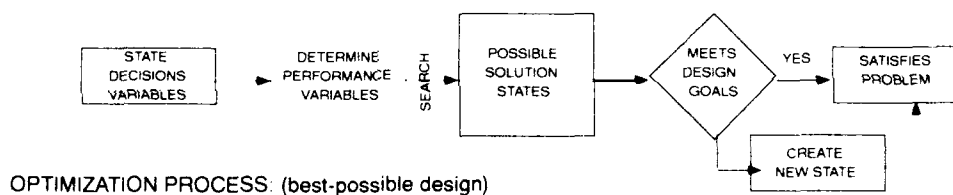
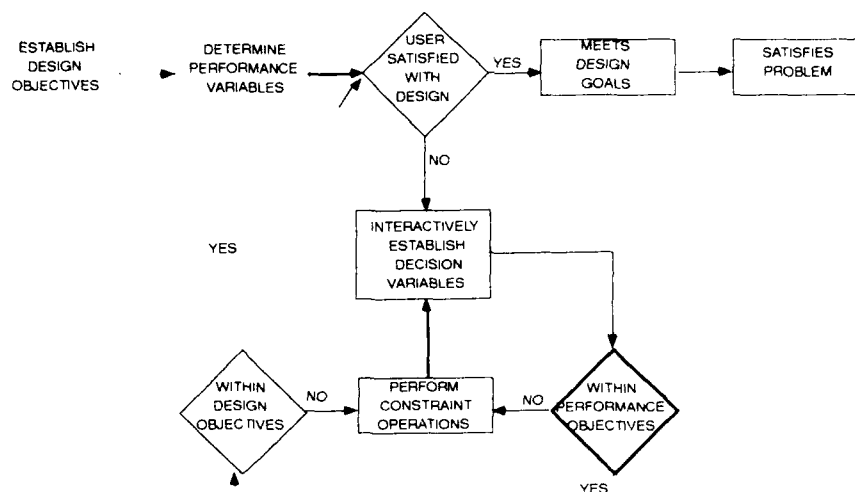


Figure 7. Flow Diagram of the optimization process of achieving problem satisfaction.

The implementation of the Constraint-Based Modeler, chooses to model the design process, not as a complete generator of solutions, but as an interactive tool in which the designer can manipulate the proposed solution within the process to guide the solution to a satisfactory conclusion. This method establishes the criteria in similar fashion to the optimization model, that is prior to the introduction of a design solution. However, in contrast, it does not search through the possible states, it limits or constrains the interactive generative process to that of the design knowledge established.



INTERACTIVE CONSTRAINT PROCESS: (solution generated within constraint limitations)

Figure 8. Flow diagram of the interactive constraint process of achieving problem satisfaction

The implication of this approach to the current CAAD theory centers around the interactive use of the design knowledge during the conceptual stage of design. The application seeks to utilize this information interactively in a graphic and visual mode, introducing design knowledge, and therefore constraints, to a three-dimensional solid modeler.

1.3 Historical Basis

The study and application of design knowledge to Computer-Aided Architectural Design is not a new concept. The historical basis of this thesis relies on the foundations set forth by many theories proposed over the past thirty years. Most notable are the foundations provided in three main areas: empirical studies, theoretical issues, and education. The area of empirical studies, (Eastman, 1970; Foz, 1973; Henrion, 1974; Krauss and Myer, 1970; Negroponte, 1970), focused on the development of models that account for the behaviors of architectural practice, with specific emphasis on formalizing the design process. Theoretical issues (Coyne, 1990; Freeman and Newell, 1971; Mitchell, 1990; Reitman, 1964; Simon,

1973; Yessios, 1987), focused on the development of a priori paradigms for design. Major emphasis in this theoretical aspect was in representation of knowledge. The third area, education, (Broadbent, 1973; Hanks, 1977; Jones, 1979; Wade, 1977) focused on the development and advancement of design methodology, with emphasis on defining design and how its related knowledge is used and how it should be practiced (Akin, 1986).

These studies pioneered the area of knowledge-based applications to CAAD, and individually, provide a valuable contribution to the study of this field. Therefore, it is essential that the development of a constraint-based modeler with foundations of knowledge application take in consideration these studies. The implementation of a constraint-based solid modeler, builds on those contributions in an effort to provide a functional, and workable, application of design knowledge to aid in the CAAD development.

1.4 Goals and Expectations

The application of design knowledge to *the interactive modeling process* as illustrated and presented, takes into account the work of early and current pioneers in the area of CAAD. The goal of this thesis was to develop a constraint-based solid modeler for architectural applications which can be utilized as a tool for Computer-Aided Architectural Design by allowing for the definition and implementation of specific design knowledge which constrains the behavior of user definable three-dimensional geometric entities. The development of such a modeler was to be an extension to the existing educational solid modeling program, MacMod844, developed, and provided by the Department of Architecture, The Ohio State University. The constraint-based solid modeler was expected to achieve four main objectives, which when implemented, would achieve these goals.

The first objective, was to provide the ability to generate, represent, and manipulate three-dimensional geometric entities through the use of a graphical interface. This includes the facilities to store and support geometric and topological editing features at the point, segment, face, and volume levels. (Existing features of MacMod844 educational modeler.)

Secondly, the implementation was to allow for the specification of design knowledge applicable to user definable three-dimensional geometric entities such as solids or spatial representations. This included the ability to define, modify, store, and retrieve constraints upon the environment as well as the individual entities.

The third objective was to provide the ability to manipulate the three-dimensional geometric entities in a manner which is consistent with the behavioral characteristics dictated by the entity specifications. Thus, constraining the operations, scale, translate, and rotate, to the design knowledge specified for the entity.

Finally, the last objective was to provide the ability to extract information from an entity, which has been provided by the specification of the entity, and the derivation of information from the specifications and geometrical data.

The primary expectation of this thesis is to contribute towards the development of an architectural solid modeler, which has the ability to represent information about a specific entity, as a foundation for design research, education, and practice. Four main goals are expected to be attained in this research. One, to support the theoretical foundations which have preceded this research. Two, to delineate and define the components of a constraint-based solid modeler, including the representation of knowledge, problem solving process, and the interaction between the designer and the modeler. Three, to implement and evaluate such a modeler. And four, to provide a foundation for further research and education in the use of knowledge-based systems in CAAD applications.

By achieving the four main goals set forth, the constraint-based solid modeler for architectural applications provides a designer with the ability to utilize and apply a small domain of design knowledge interactively in the generation and manipulation of architectural elements, thereby, providing a dynamic and interactive modeling environment which is behaviorally constrained.

The remainder of this thesis explores and illustrates the functional constraint-based operations which were implemented in the modeler, a system overview and user's manual to

C•Mod, the application of the constraint-based solid modeler, the internal workings and algorithms used to implement the constraint operations, applications of the working program, extensions and future directions to be explored, and finally conclusions surmised from the research and implementation.

CHAPTER II

CONSTRAINT-BASED MODELING OPERATIONS

The realistic generation and manipulation of solids and spaces in an interactive environment requires the use of specific information pertaining to the entities themselves and to the interactive behavioral attributes between entities. To facilitate this realistic interaction of entities, two types of information are required, physical and relational attributes. Physical attributes such as representation, dimensional data, area limitations, volumetric requirements, rotational attributes and spatial mobility, are applied upon the entity itself. Relational attributes such as proximity criteria, containment characteristics, and associative attributes, are applied to entities which interact with one another. By applying this specific information, the interactive solid modeling process is constrained and limited to the bounds established. The results of this application of specific information or knowledge about an entity in an interactive solid modeling environment is a constraint-based solid modeler.

Allowing for the definition and implementation of specific design knowledge which constrains the behavior of user definable three-dimensional entities is critical to a successful implementation of constraint-based solid modeling. A constraint-based solid modeler must, therefore, provide four primary operational capabilities: 1) The ability to specify the design knowledge applicable to user definable three-dimensional entities such as solid and spatial entities. 2) The ability to modify, store, and retrieve the entity specifications provided through knowledge specification. 3) The ability to manipulate the three-dimensional entity in a manner which is consistent with the behavioral characteristics dictated by the entity specifications. 4) The ability to extract, or query, information from an entity which has been provided by the

specifications of that entity. This chapter explores each of these primary operations in order to illustrate the role each must play in the interactive modeling process.

2.1 Knowledge Specification

The foremost important ability of a constraint-based solid modeler is to specify and distinguish specific information pertaining to an entity. Information, or design knowledge which is typically represented graphically, as illustrated in the figures below, must be specified by the user, or read from a data base containing the required information, and made available to the modeler. An interactive method of communication between the user and the system allows the user to input specific design knowledge applicable to a specific entity, or to an environment which is to be modeled. The modeler, once the design knowledge is present, can utilize this information to aid in the process of interactive modeling by providing the lower level satisfaction checking to ensure that the requirements set forth are met.

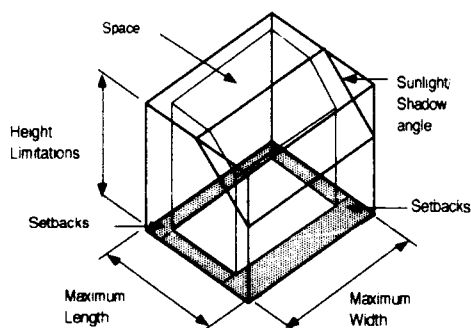


Figure 9. Design knowledge of a space.

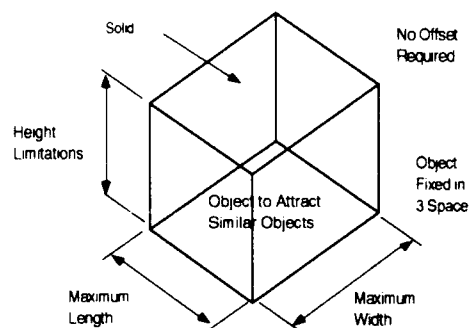


Figure 10. Design knowledge of a solid.

2.2 Knowledge Manipulation

The specification of information, or design knowledge, is the initial step in conveying the scenario in which the modeling environment is to be constructed. Once the specific entity

design knowledge has been provided, a constraint-based solid modeler must provide the ability to manipulate that information. This includes the ability to modify, store, and retrieve the entity specifications which have been specified. By allowing the user to modify constraints, or information particular to a specific entity, the environment can remain dynamic in nature. This flexibility provides a method of constraint resolution in which the user can choose to apply or disregard a particular constraint during the interactive modeling process. Additionally, by allowing a set of constraints to be stored and retrieved, a preset description of entities, or entity types, can be applied to the modeling environment without having to specify the entire set of constraints again (figure 11 and 12.)

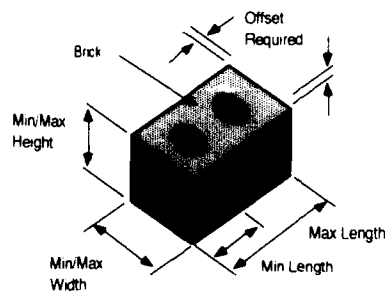


Figure 11. Constraints upon a single entity and stored as a brick definition.

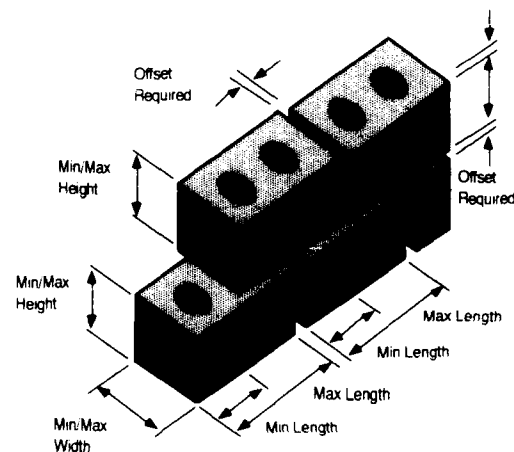


Figure 12. Preset constraints applied to all subsequently generated entities.

To allow for this flexibility, two types of constraint knowledge are to be represented, entity specifications, and system specifications. Entity specification is the set of constraints/attributes which pertain to a specific entity. It is that set of specifications which control the operations involving that entity. The second form is system specifications, or system defaults. This set of constraints/attributes are what the system uses when a new entity

is created. Both sets are accessible to the user at any given time. By allowing this manipulation of information, the user has the flexibility to control knowledge about a specific entity and the system specifications.

2.3 Constraint Modeling

The purpose of a constraint-based solid modeler is to provide an environment in which the designer can interactively generate and manipulate solid models for architectural applications which follow behavioral patterns established by the specification of design knowledge as constraints. Additionally, once the constraints have been established, the user should be free to explore possible solutions within the constraint limitations, without having to constantly check for compliance with a particular attribute or value. To accomplish this, three levels of constraint modeling are required to be provided: entity , relational , and identification. The entity level applies to the physical characteristics of the entity and relates to the generation and editing of solid objects. *Interactive relationships between entities make up the relational level, and apply to the geometrical transformations of translation, rotation, and scale.* The final level of identification is required to distinguish the differences between entities, and provides the ability to store and retrieve the constraint set. The following discussion elaborates on each of these levels, and includes an illustration of the major attributes which provide the ability to interactively model constraints.

2.3.1 Entity Characteristics

Entity characteristics are the specific physical attributes which the entity is required to satisfy during the generation and manipulation of the entity in the three-dimensional modeling process. The physical attributes provide a realistic representation of a narrow spectrum of design knowledge upon a specific entity. For the purpose of illustrating the constraint-based solid modeling process, five major physical attributes have been chosen: spatial representation,

dimensional attributes, spatial characteristics, rotational attributes, and mobility characteristics. Although an unlimited number of descriptive information can be obtained and applied to the generation and manipulation of geometric entities, these five physical attributes represent a selection of primary characteristics which govern the creation of physical objects. The following is a discussion of each of the five physical attributes.

2.3.1.1 Spatial Representation

The fundamental object description/constraint is found in the spatial representation attribute. By denoting an object as a solid entity, the modeler must perform operations on that entity as a solid. Thus, a solid is expected to behave as a solid entity, and a spatial entity is expected to behave as a spatial container of other entities. The distinction between the two representations lies in the ability to contain mass. Solid elements are just that - solid - they contain mass, and cannot be occupied by another entity containing mass. Spatial elements do not contain mass, they are composed of space, or a spatial void, and have the ability to contain

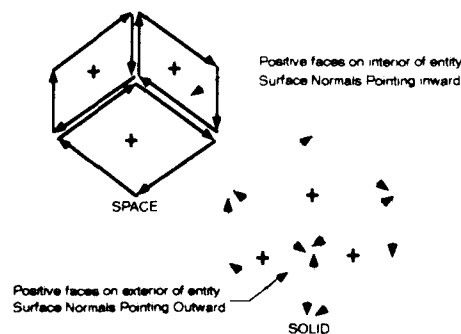


Figure 13. Representation of a solid entity and a space entity.

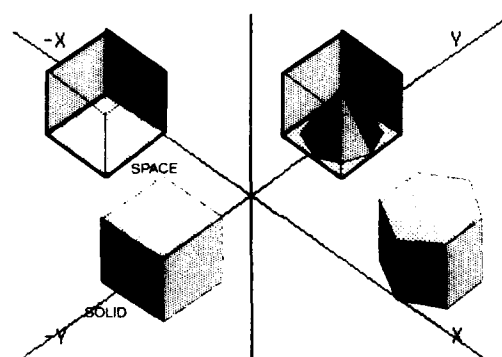


Figure 14. Three-dimensional representation of solids and spaces.

other entities such as spaces or solids. Figures 13 and 14 illustrate the physical and visual differences between a solid and spatial entities. As a fundamental characteristic of constraint-

based solid modeling, this ability to distinguish between spaces and solids interactively provides a strong foundation to utilizing design knowledge within an interactive environment which allows immediate feedback to entity interaction.

2.3.1.2 Dimensional Attributes

Dimensional attributes constrain the generation of entities, specifically extrusion, as well as the geometric editing features of the modeler. The application of dimensional data must be capable of supporting editing of all topological levels: point, segment, face, and object. To facilitate this characteristics in an interactive environment, a bounded box dimensional restriction upon the generation and manipulation of the entity must be provided (figure 16). The bounding box method maintains the dimensional information regardless of the effects of 3D transformations such as scale, rotation, or translation, and regardless of the topological level on which the operation is being conducted (figure 17). Retaining the dimensional data in this way allows the user to specify the minimum, maximum, and incremental dimensional constraints upon a geometric entity. Each of these values may be set and activated for the width, height, and length attributes independently or in combination. Figure 15 illustrates the minimum and maximum values establish a range of allowable values to be used during the modeling process.

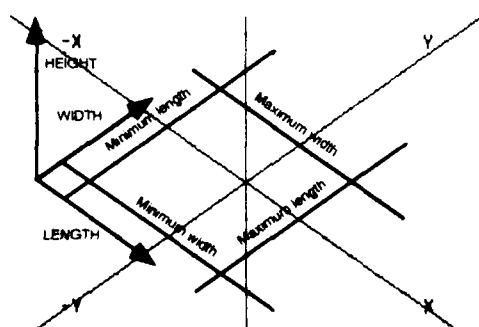


Figure 15. Visualization of the width, length, and height dimensional criteria.

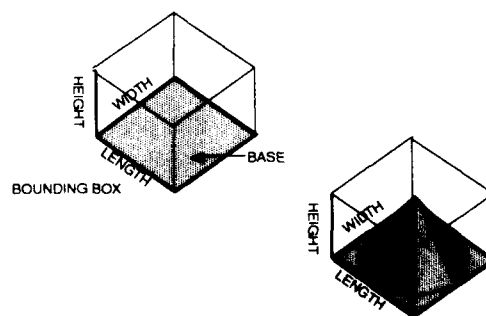


Figure 16. Three-dimensional representation of the bounding box and dimensional data.

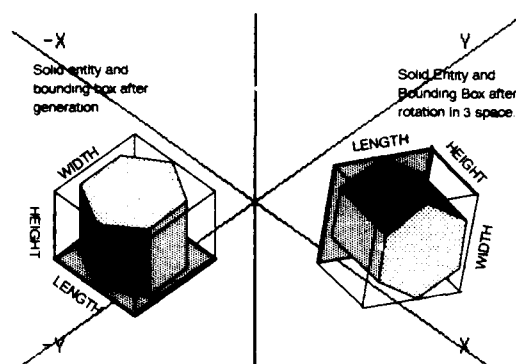


Figure 17. Bounding box and solid entity in three space.

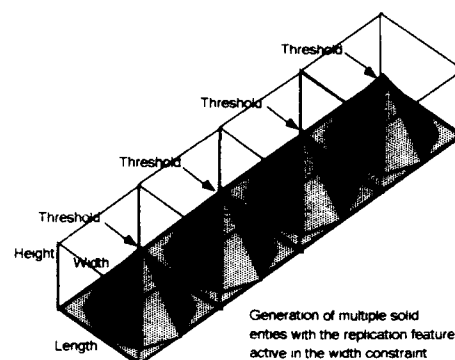


Figure 18. Replication of a solid entity in the width dimension.

Simply restricting the operations to maximum dimensional limitations does not fully capitalize on the nature of dimensional data. Modular construction, such as kitchen cabinetry, utilizes dimensional data to indicate when another module should be generated. The cabinets, for example, would split or replicate when they are extended beyond a particular dimensional limitation. To support this type of design knowledge, the modeler must allow the user to specify a replication threshold, and replicate the entity when generation or manipulation extends beyond this threshold. With the replication feature disabled, the generation and manipulation of the entity is constrained to the limits imposed by the minimum and maximum dimensions provided. Incremental rubber-banding, scaling, and translation will be snapped to the increment value provided. When the replication feature is enabled, however, the maximum value becomes a threshold value used to invoke replication. Extending beyond this maximum value will result in the creation of an additional element with the same constraints and attributes, thus replicating the entity (figure 18).

2.3.1.3 Spatial Characteristics

Spatial characteristics impose similar constraints to the generation and editing features of the modeler as do dimensional attributes. In addition to, or in lieu of, dimensional

attributes, spatial constraints of area and volume may be required. Providing spatial constraints would limit, or restrict, the generation and manipulation of the entity based upon minimum and maximum square and cubic units of measure, thereby allowing the definition of room size in square units, or spatial size in cubic units (figures 19 and 20). A minimum value would establish the smallest area or volume that entity may possess, and require the modeling process to maintain this minimum value. A maximum value would establish the largest area or volume that entity may possess, and would be used to limit geometric editing operations upon that entity. Since most applications of spatial criteria utilize an orthogonal configuration, the modeler must provide an option which allows the user to specify whether or not to constrain the entity to this orthogonal continuity during transformations. A constraint-based solid modeler could therefore allow interactive solid modeling of programmatic requirements maintaining this information during the process.

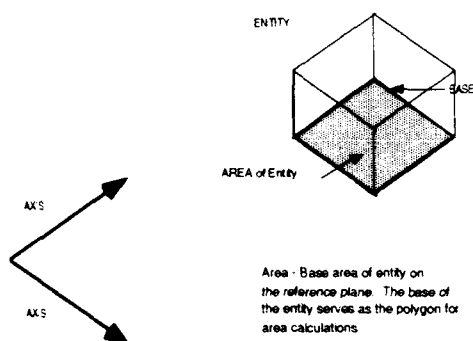


Figure 19. An illustration of the area characteristic.

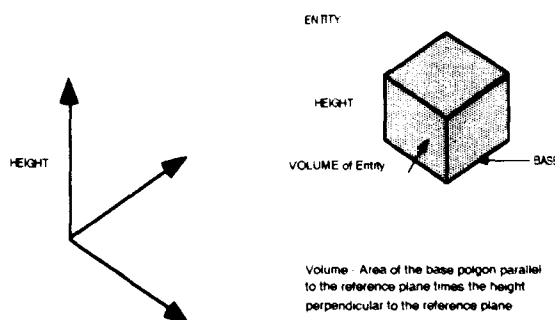


Figure 20. An illustration of the volume characteristic.

2.3.1.4 Rotational Attributes

Rotational attributes effect the geometric editing feature of rotation. This becomes important when one considers, for example, the placement of entities which can only be configured at right angles within a space. By constraining the rotational level and degree of

rotation which can be imposed during the geometric editing operations, these restrictions can be represented in the interactive modeler. In order to facilitate this constraint two types of specification must be allowed, level, or axis of rotation, and increment, the degree of rotation allowed. Specification of the level of rotation would provide the selection and indication of allowable rotation in three-space, and would include activation and limitation of each of the three major axis. Indication of the rotation increment, in terms of minimum and maximum values, would provide the allowable range of rotation from 0 to 360 degrees in right-handed space (figure 21). In addition, the rotational increment would constrain the incremental rotation about the selected axis. Thus, an entity can be restricted to z axis rotation at 45 degree increments (figure 22). Implementing this characteristic would provide an environment in which entities can be constrained or limited to specified locations or configurations.

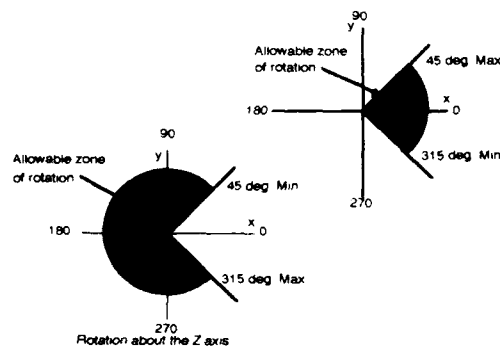


Figure 21. Visualization of the limits on the x-y reference plane.

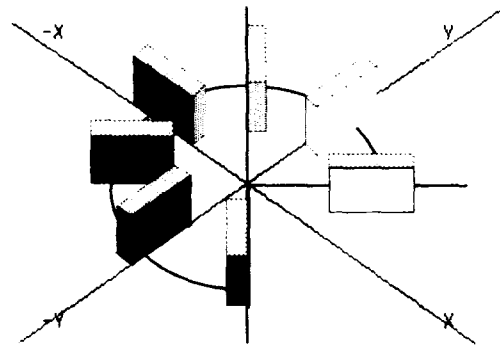


Figure 22. Example of an object rotated about the origin, constrained to limits.

2.3.1.5 Mobility Characteristic

Mobility characteristic's establish the degree of interaction the element will exhibit when acted upon by another element and provides the ability of rigid placement within three-dimensional space. A primary use of the mobility characteristic is to fix the location of an entity so that the interaction between other elements will not cause the entity to translate from

its position. Once an object is placed in the modeling environment, it may become necessary to fix, or freeze, its position, as in the placement of a wall, stair, column, or any construction component. A fixed entity therefore becomes an obstacle which, when interacted upon by other entities, will restrict, or limit their movement (figure 23).

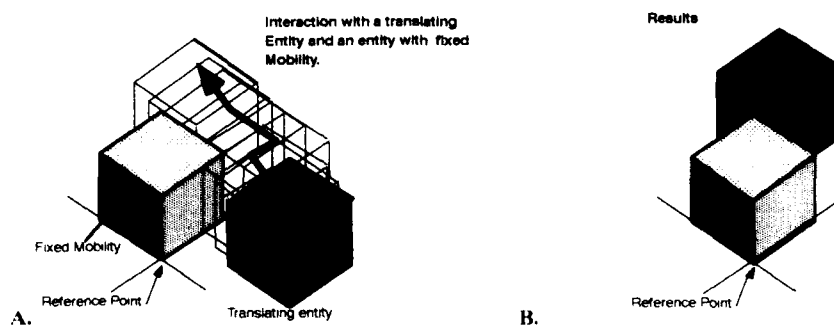


Figure 23. Illustration of an interaction between a translating entity and an entity with fixed mobility. A. Indicates the position prior to translation, illustrating the path of travel. Note the the object must travel around the object. B. Indicates the result of the interaction.

An entity with a mobility characteristic of free, however, is not confined or restricted to a particular location in three-space. Free mobility enables the entity to respond to the interaction of other elements in accordance with the expected behavioral patterns established through the specification of other constraints (figure 24). The mobility characteristic does not effect any geometric or topological generation or editing feature directly. It does, however, effect the resultant activity from associative relationships and manipulation of other entities which will be discussed in section 2.3.2.3.

Used in combination with the spatial representation characteristic, the mobility characteristic is the second most important feature an interactive constraint-based modeler must possess. These two characteristics alone can model, through direct response in real time, the interaction of solid objects in three-dimensional space.

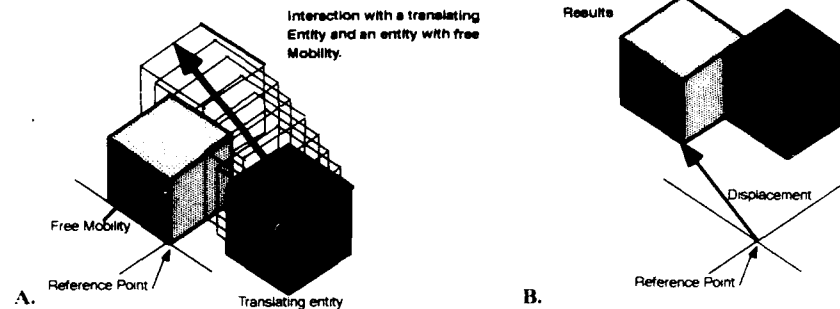


Figure 24. Illustration of an interaction between a translating entity and an entity with free mobility. A. Indicates the position prior to translation, illustrating the path of travel. B. Indicates the result of the interaction.

2.3.2 Relational Characteristics

Relational characteristics are attributes an entity possess which indicate how that entity is to interact with other entities. Three relational attributes, proximity relationships, containment relationships, and associative relationships, used separately, or in combination, are required in order to provide additional behavioral constraints upon the manipulation of the entity within the modeling environment. The following is a discussion of each of the three relational characteristics.

2.3.2.1 Proximity Relationships

Proximity relationships establish a zone around, for solid entities, and within, for spatial entities, the entity which provides and acts as a buffer, or clear zone between the entity and any other entity. This zone provides the minimum distance in which other entities may encroach upon, as well as the distance criteria for proximity detection and activation of associative responses (refer to section 2.3.2.3). A buffer zone is created by establishing an offset from the bounding box based on the width, length, and height distance from the entity, and is to maintained regardless of the geometric transformations applied upon the entity (figure 25). Therefore, a specification of an offset along a particular edge, such as the width edge,

must be maintained throughout the modeling process. The use of a buffer zone, or offset, allows the modeler to represent non-physical entities such as distances between floors, or can act as physical representations such as mortar between brick entities. A modeler can therefore act as an abstract representation of design elements and can provide the ability to use design knowledge in an early stage of the design process.

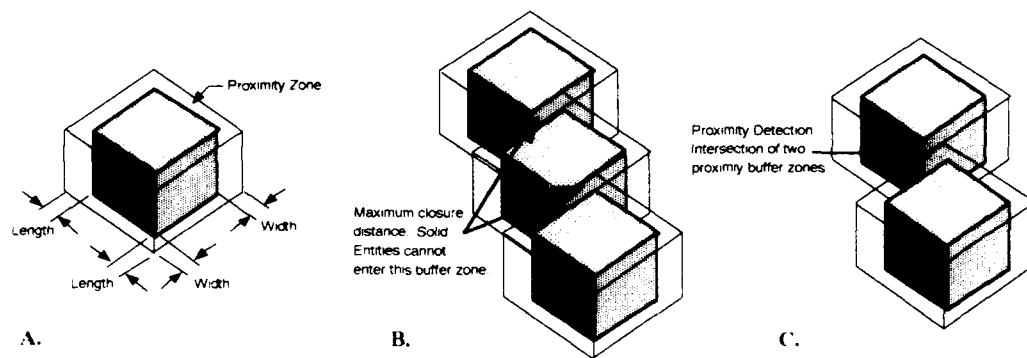


Figure 25. An example of the proximity zone and it's relationship with the entity. A. Illustrates the buffer zone around the entity. B. Illustrates the offset created between three entities. C. illustrates when proximity detection occurs.

2.3.2.2 Containment Relationships

Containment establishes the relationship between dissimilar spatial representations. A hierarchical relationship can establish constraints which effect the extrusion, scale, rotation, and translation of entities, and will restrict such operations on subordinate entities to the limits of the bounds of the spatial entity (figure 26). This has the effect of enclosing entities within spaces, such as the design space illustrated in chapter I, and effectively constraining the interaction and manipulation of entities within that space. Depending upon the spatial representation given to an object, the modeler should provide several options to the user. These options must include no containment, solid contained by a specified volume, or a spatial

entity which contains user specified solid entities. No containment allows the entity to be non restricted in any space. If containment is desired, the entity must belong to a spatial entity, and thus contained by that entity, or contain solid entities, depending on the spatial representation. It is important to note that in the event that a solid entity has no containment relationship between a spatial entity, the solid entity should have complete freedom to penetrate or exceed the bounds of the spatial entity, and thus is not be restricted or contained by the spatial entity.

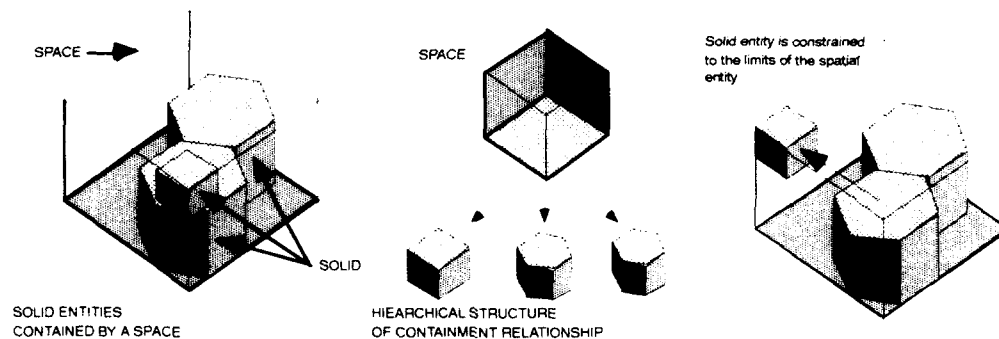


Figure 26. Example of the containment relationship and it's interaction between entities.

2.3.2.3 Associative Relationships

Associative relationship establish the response an entity will exhibit during the interaction between itself and another entity. Utilizing an offset tolerance from the proximity zones for each entity, the interaction between two or more entities is affected by their associative relationship indicated, and respond in one of three actions, no response, repel or back away, or attract and attach. No relationship would indicate that there is no inherent association required for that entity. A repelling relationship would indicate that the object will oppose, or repel another entity encroaching within the tolerance. Attracting relationship indicates that the object will attract, or try to attach itself, to another object encroaching within the tolerance.

The associative relationship utilizes the mobility characteristic of the secondary entity, or the entity which is being acted upon, as a basis to determine the type of response the entity is to exhibit. If no associative relationship between the entities is desired, the entities must behave in a manner which is consistent with solid objects. However, if an association is established, and depending upon the mobility definition of the secondary entity, the associative response will be enacted by the interacting entity or by the secondary entity. The associative relationships effect all transformations, scale, rotation, and translation, interactively, and must

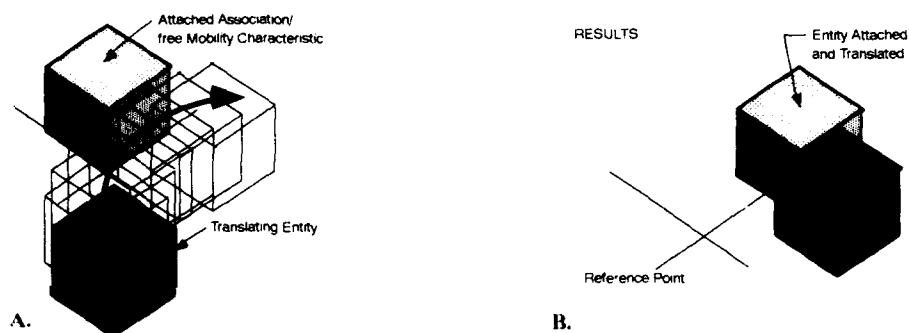


Figure 27. Example of the attract association and secondary entity with free mobility. A. Indicates the position prior to translation, illustrating the path of travel. B. Indicates the result of the interaction.

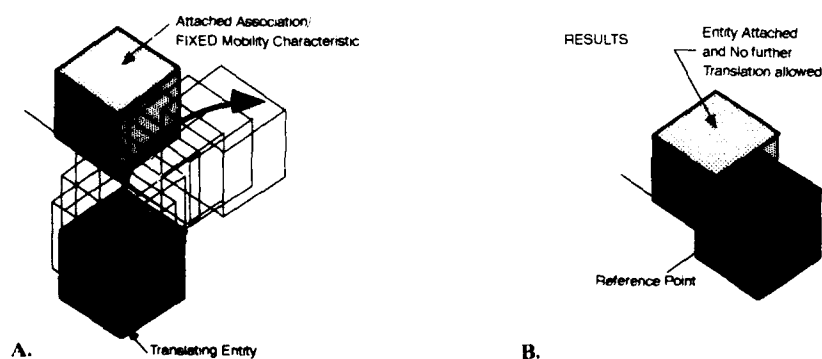


Figure 28. Example of the attract association and secondary entity with fixed mobility. A. Indicates the position prior to translation, illustrating the path of travel. B. Indicates the result of the interaction.

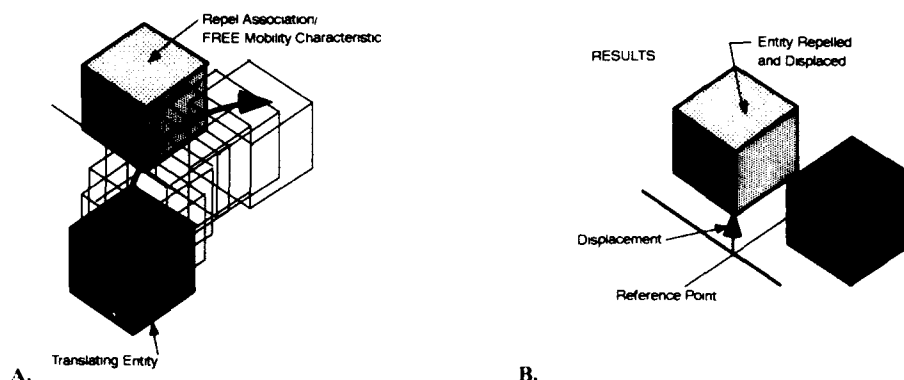


Figure 29. Example of the repel association and secondary entity with free mobility. A. Indicates the position prior to translation, illustrating the path of travel. B. Indicates the result of the interaction.

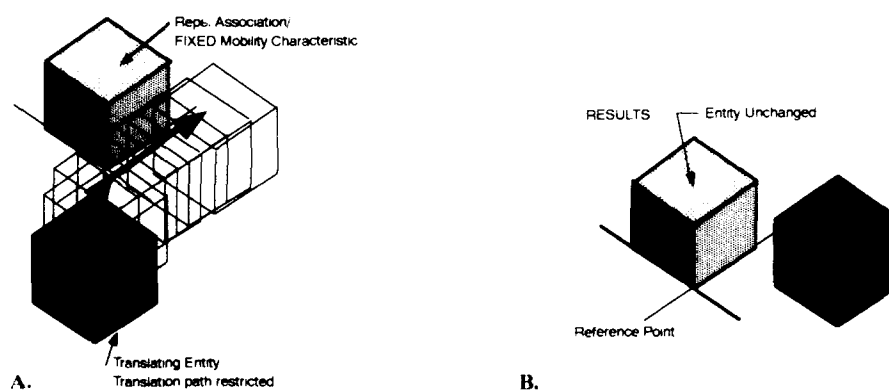


Figure 30. Example of the repel association and secondary entity with fixed mobility. A. Indicates the position prior to translation, illustrating the path of travel. B. Indicates the result of the interaction.

provide an immediate visual feedback to the user during the modeling process. When any of these operations are performed, and the resulting activity causes the entity to encroach upon the tolerance of another entity, the entity will behave accordingly, depending upon the mobility characteristic of the entity. Thus an entity with an associative relationship of attract, meets a fixed entity, the entity will snap to the edge of the secondary entity. In the event that the secondary entity was free, it would snap to the edge of the primary entity (figures 27 and 28).

For the repel association, an entity which meets a fixed entity will be repelled, or pushed away from the secondary entity. Conversely, if the secondary entity was free, it would be repelled, or pushed away from the primary entity (figures 29 and 30).

2.3.3 Identification Characteristics

The identification attribute allows the unique identification of a set of constraints which have been established for a given entity. This type definition attribute is required when storing and retrieving constraint information.

2.3.3.1 Type Definition

This is a unique identifiable descriptive attribute associated to the set of constraints representing an entity. The specification of a type definition to a specific set of constraints provides the capability to store and retrieve the entire set of constraints with a single identifying macro (figure 31). By allowing the set of constraints to be identified as a single type definition, the set of constraints can be created and stored once, and then retrieved and assigned to entities with a single reference indicating that type. This ability to assign a complete set of constraints with a single reference allows quick and efficient use of the design knowledge about entity types without specific knowledge about all the particular attributes required to represent that type.

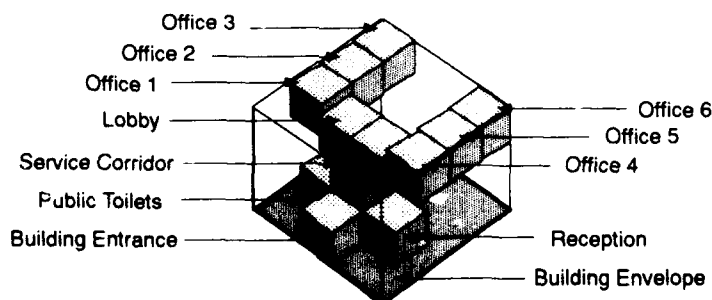


Figure 31. An illustration of the type definitions.

2.4 Knowledge Query

The last of the primary operations involve the ability to extract or query the modeler for information pertaining to a specific entity. This specific knowledge pertaining to an entity must be supported in two forms, defined and derived. Defined information is specific knowledge pertaining to an entity which has been defined or stored as a set of constraints. This type of information should be readily available in the interactive modeling process and accessible without excessive disturbance to the process. The second type, derived, should allow the user to retrieve specific knowledge about the entity which has been derived, or determined from the design knowledge specified and the geometric data of the entity. Information pertaining to derived knowledge includes square footage, volumetric data, actual dimensional data, and location data. This form of communication between the user and the system would allow immediate checking and verification of the design knowledge relating to any specific entity in the modeling environment. In addition, it allows the user to question the response of the constraint-based solid modeler, and determine why a particular action was taken.

2.5 Direction

From the discussion and illustration of the primary operations which should encompass a constraint-based solid modeler, it is evident the use of design knowledge in combinations, or separately, provide the designer with a valuable tool in the design process. This tool can provide the user with the ability to generate and manipulate design knowledge to dynamically constrain, with realistic information and results, the process of solid modeling. The interactive generation and manipulation of solid elements within design knowledge will allow an efficient and expedient method of spatial, or mass, modeling and would be able to provide a powerful foundation to schematic and conceptual design.

The next chapter introduces C•Mod, an prototypical interactive constraint-based solid modeler which provides a demonstration of the viability of many of the concepts presented.

CHAPTER III

C•MOD - SYSTEM OVERVIEW AND USER'S MANUAL

The prototypical constraint-based solid modeler, C•Mod, was written and developed as an extension to the MacMod844 shell provided by the Department of Architecture, The Ohio State University, and currently runs on the Apple® Macintosh® platform. C•Mod provides an environment in which the designer can interactively generate and manipulate solid models for architectural applications which follow behavioral patterns established by the specification of design knowledge as constraints. Additionally, once the constraints have been established, the

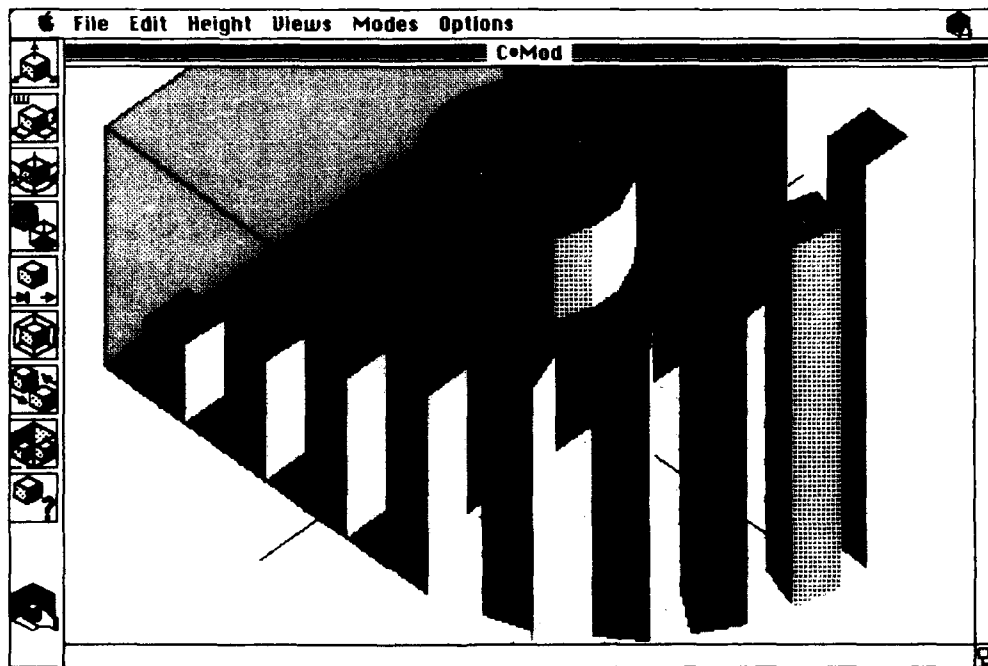


Figure 32. C•Mod prototypical application.

user is free to explore possible solutions within the constraint limitations, without having to constantly check for compliance with a particular attribute or value. The following chapter includes an introduction to the C•Mod prototypical application, an overview of the menu and command structure, a brief discussion of the basic capabilities of the solid modeler, and a full description of the constraint-based operations. The goal of this chapter is to provide the reader with the interface, characteristics, and use of C•Mod.

3.1 Introduction to C•Mod

C•Mod is an interactive constraint-based solid modeler which predominantly communicates with the designer in a manner which is consistent with the way most designers communicate, graphically. A user has the ability to generate and manipulate solid elements graphically, with the use of the mouse, and can visualize the results interactively. This section introduces C•Mod, discusses how to activate the program, and illustrates the navigation through the menus, icons, and windows utilized by the application.

3.1.1 Activating the program

C•Mod utilizes the basic method of program activation suitable for the Apple® Macintosh® operating system. It is assumed that the reader is familiar with this operating system, and is comfortable with the point, click, and drag method of mouse communication. The program supports three icons used to represent specific information; application icon, project files, and constraint types (figure 33). C•Mod is started by double clicking the application icon.



Figure 33. C•Mod icons. A) C•Mod application, B) project files, C) constraint types.

3.1.2 Graphical Environment

Upon start-up, the program will display the graphical environment and is ready to begin a session. Since there are a number of unique features to C•Mod, this section explains the graphical environment and discusses the navigation of the screen.

C•Mod consists of four basic parts, a menu bar, a graphic window, a message window, and a tool box (figure 34). The menu bar provides access to the features and operations of the program. The graphic window provides the visual communication between the user and the application. It consists of a reference plane, which serves as a platform, or drawing surface, and the three major axis, which serve as Cartesian coordinate references to three-dimensional space. The third basic element is the message window used to communicate textual information. The current cursor location is indicated in this window. The last major element is the tool box. The tool box provides access to specific features of the application based on the current active mode of operation.

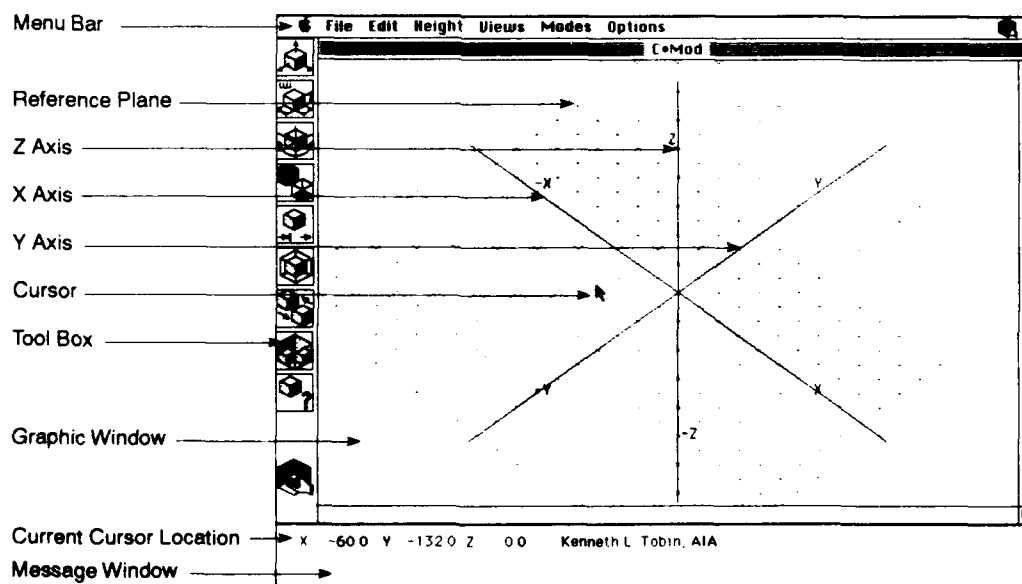


Figure 34. C•Mod graphic environment.

3.2 Overview of the Menu and Command Structure

The primary form of communicating instructions, or commands, to the program is facilitated through the use of the menu bar (figure 35). The menu bar provides seven pull down menus with which the user can select the various options and modes of operation available. The pull down menus include the apple, file, edit, height, views, modes and an options menus. An overview of each of the menu items is presented to briefly introduce the user interface, and to illustrate the options or modes which are available under each menu.



Figure 35: The C-Mod menu bar.

3.2.1 The Apple Menu

The Apple menu contains the **About Constraint Mod...** command which displays a dialog box with a brief description of the C-Mod application. This dialog box is purely informational, and does not serve any other purpose in the modeler. To dismiss this dialog box, simply click anywhere within the box. In addition, this menu allows the user to utilize the mutlifinder, if active, to access other programs, and allows access to desk accessories which are currently loaded in the systems folder. Figure 36 illustrates a cut-away portion of the apple menu. Figure 37 illustrates the dialog box which is displayed when About Constraint Mod is selected.

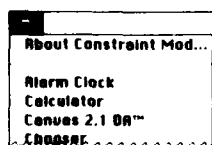


Figure 36: Apple Menu.

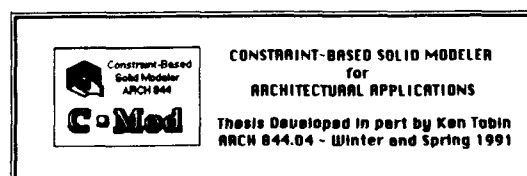


Figure 37: About Constraint Mod dialog box.

3.2.2 The File Menu

The **File** menu provides general file processing operations for the application (figure 38). These include the ability to retrieve and store the project files, as well as quitting the application. As previously mentioned, there are two file types used by C•Mod, project files and constraint type files (figure 39). The file menu provides access only to the project files. This is important to note when storing information. Project files contain all information pertaining to a current scene, or environment. Whereas, only constraint type definitions are stored in the constraint type files (refer to section 3.4).

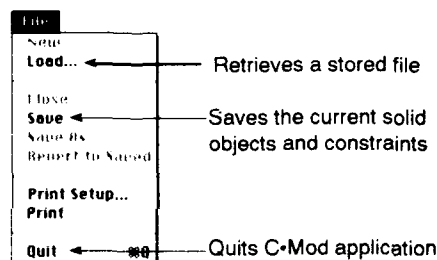


Figure 38: File Menu.



Figure 39: File icon types.

The **Save** item activates the standard file dialog box (figure 40), which prompts the user for a file name. In the event that the name specified matches a file which exists, the user will be prompted to change the name or rewriting over the existing file. The file saved contains all of the information required to create the scene, including the object data structures and the constraint data structures.

The **Load** item activates the standard file dialog box (figure 41), which prompts the user to select a file from the listing of available files. The user can select the desired file by double-clicking on the file name, or by clicking once on the name and selecting open. In the event the file desired is not found in the current folder, additional folders can be view by

selecting the top file descriptor and dragging down the list of hierarchical files. Access in this mode is limited to files created only by the save item above while in this application.

The **Quit** item terminates the program. Note that quitting the program prior to a save will not prompt for a decision to save or not. The program does not automatically save the current set of data.

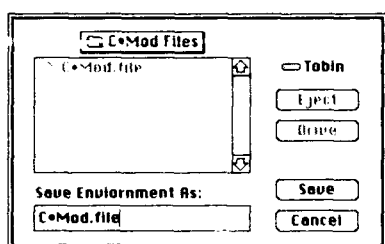


Figure 40: Save file dialog box.

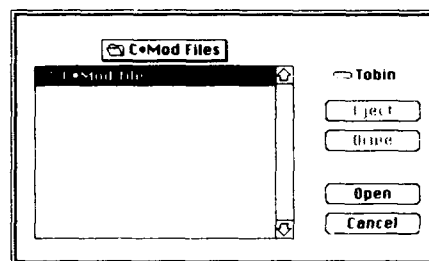


Figure 41: Load file dialog box.

3.2.3 The Edit Menu

The **Edit** menu provides the means of controlling how the modeler is to perform in three-dimensional space (figure 42). The primary means for this is through the selection of a reference plane which is used during the generation and manipulation of objects. In addition, the edit menu allows the user to clear the entire screen to begin a new session. The selection of **Clear Mem** will erase all items from the graphics window. Note that the undo command is not active, and will not perform any operations.

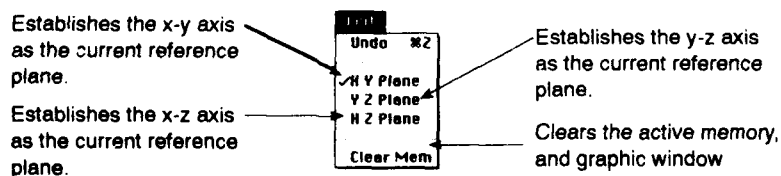


Figure 42: Edit Menu.

The items **X-Y Plane**, **Y-Z Plane**, and **X-Z Plane**, set the current active reference plane accordingly. This is the controlling plane of reference for the generation, and manipulation of entities within a three-dimensional space. In other words, if the active plane is set to x-y, drawing a line in the three-dimensional environment will take place on the x-y plane. This becomes important during geometric operations such as translation, which will perform the operations parallel the active reference plane. To aid the user in distinguishing the which plane is active, C•Mod displays a yellow grid indicating the plane of reference (figure 43).

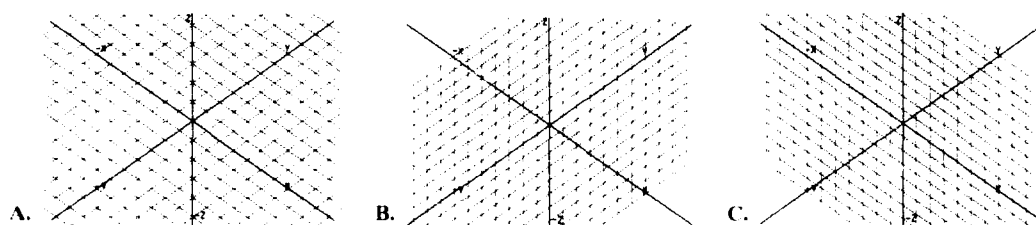


Figure 43: An illustration of the reference planes viewed from 45-45 angle. A) Illustrates the x-y plane. B) Illustrates the y-z plane. C) Illustrates the x-z plane.

3.2.4 The Height Menu

The **Height** menu provides a means of establishing the height, or the perpendicular distance from the reference plane (figure 44). This menu provides the specific height value in support of generation operations. The values, ranging from 24 units to 240, establish a fixed positive distance, and are consistent with the grid spacing indicated on the reference plane.

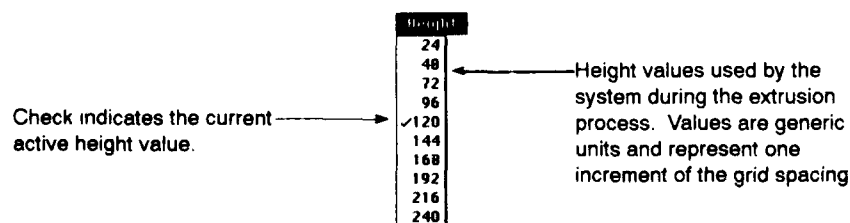


Figure 44: The Height Menu.

3.2.5 The Views Menu

The **Views** menu enables the user to view the three-dimensional modeling world from a variety of positions (figure 45). By selecting the desired view angle, the application will regenerate the scene from the specified angle, and will allow interactive modeling from that perspective. The menu provides three methods of viewing the scene, as an axonometric projection, as a plan, or as an elevation.

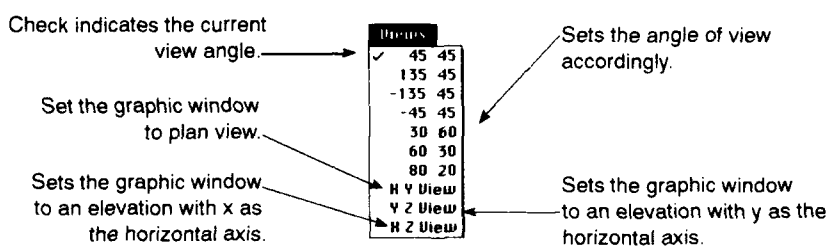


Figure 45: The Views Menu.

The axonometric views are the first seven selections in the menu, and are indicated by the numerical descriptions. The values stand for the degree of rotation about the z-axis and x-axis respectively. To obtain a plan view, the selection of the x-y view will provide a projection of the scene with a line of sight along the z-axis. The elevations are viewed by selecting the y-z view or x-z view. Figure 46 illustrates various views of the reference plane.

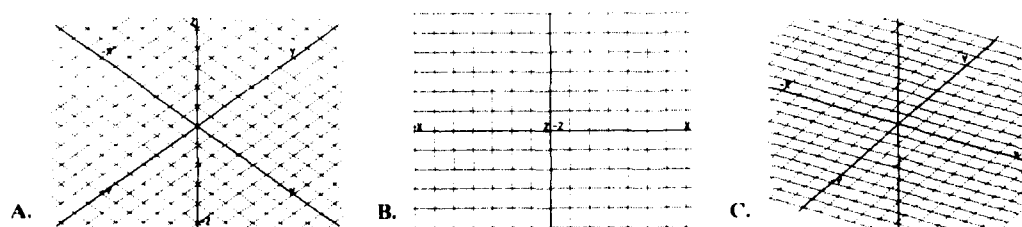


Figure 46: The effects of the view command on the reference plane. A) 45-45 axonometric view, B) X-Y plan view, and C) 30-60 axonometric view.

3.2.6 The Modes Menu

The **Modes** menu provides the user a method of activating the desired operations available in C•Mod (figure 47). C•Mod currently supports ten modes of operation. These modes include two-dimensional drawing, three-dimensional extrusions, three-dimensional convergence, three-dimensional objects of revolutions, geometric editing, topological editing, texture mapping, void modeling, color surface shading, and constraint manipulation. The following is a brief discussion of each of the operation modes. In the event that more information is required, section 3.3 elaborates and illustrates each of these operations.

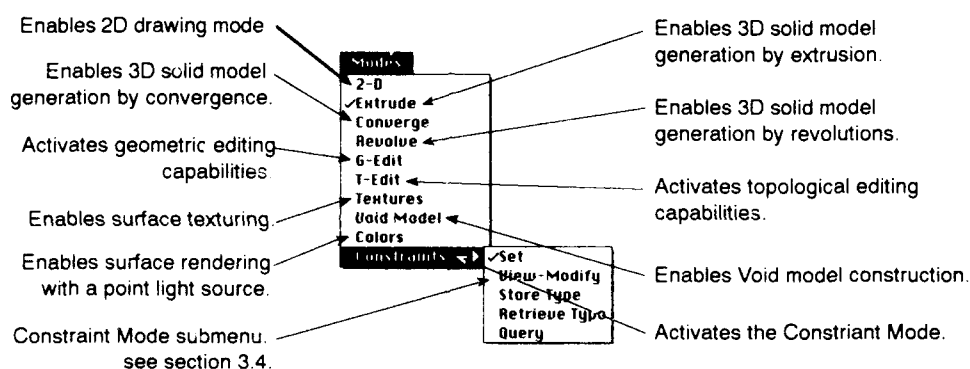


Figure 47: The Modes Menu.

The basic operations of 2-D, Extrude, Converge, and Revolve, make up the generation operations of C•Mod. With these operations the user has the ability to create two-dimensional and three-dimensional forms. The **2-D** mode of operation provides the ability to generate two-dimensional objects on the active reference plane. This mode allows line, and polygon drawing, and is used to aid in the generation of objects of revolution. The **Extrude** mode of operation provides the ability to generate three-dimensional objects from a base drawn on the active reference plane. Solid objects are generated in this mode. **Converge** is similar to the extrude mode in that it generates three-dimensional objects from a base. However, the

converge mode extrudes the object to an apex point at a height determined by the height menu. This mode allows the creation of pyramidal solid objects. **Revolve** provides a method of object generation termed objects of revolution. This mode of operation is performed on a pre-generated two-dimensional object, and will generate a solid object by sweeping the form about an axis. In this mode a sphere or torus can be generated.

The operations G-Edit, T-Edit, and Constraints, provide editing features required to control and manipulate an object in three-space. **G-Edit**, or geometric editing, enables the transformations and reformations of an entity in three-space. Editing in this mode include point, segment, face, and object translation, as well as scale and rotation at the object level. **T-Edit**, or topological editing, enables the insertion and deletion of points and segments into the object itself. **Constraints** provides the user with the ability to specify and manipulate design knowledge. Refer to section 3.4 for a complete description of this mode.

C•Mod provides two methods of rendering the object created, texture mapping, and surface shading. **Textures** allows for the application of surface textures to be applied to the object. Textures include cross-hatching, stone, brick, and stipple. In addition to textures, **Colors** provides a method of surface shading the object with a point light source. The effects of each is to provide a more accurate visual representation of the object being modeled.

A final mode of operation allows the creation of building elements from the generation of two-dimensional void representations. **Void Model** allows the generation of a plan in which doors and windows can be added. From these two-dimensional representations, the user can create three-dimensional building models, as well as solid roof objects.

3.2.7 The Options Menu

The **Options** menu selection provides various options to the user to aid in the use of the modeler (figure 48). These features aid in the drawing, visualization, and manipulation of the object in the three-dimensional environment. **Snap Line** will snap the mouse selection to

the next half increment of the reference grid. This allows direct placement of objects and points without precise specification. **Hide Back Face** will render the scene with backfaces eliminated from view. **Show Light Source** will illuminate the scene and render the objects with the appropriate surface color. Both options provide a visual enhancement to the modeling scene. **Apply Constraints** provides the user with the option to restrict the generation and manipulation of the three-dimensional environment to satisfy all constraints specified. With this option selected, the solid modeler becomes a constraint-based solid modeler. A final option, **Show Bounding Box**, is used in conjunction with the Apply Constraints option. This option renders the bounding box of the object, and color codes the edges to indicate the dimensional data used by the constraint mode.

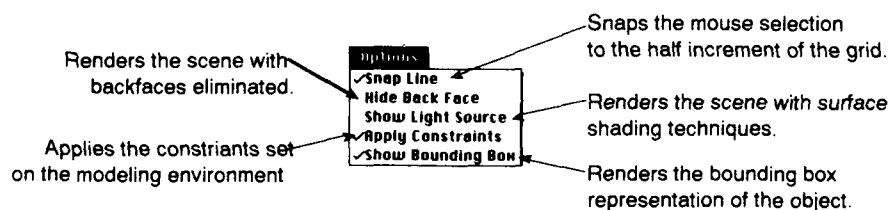


Figure 48: The Options menu.

3.3 Discussion of Basic Capabilities

To gain an understanding of the basic capabilities of C•Mod, it is appropriate to discuss each of the modes of operation in greater detail. This discussion will illustrate the basic features, concentrating on the modeling capabilities, and will provide the reader with a better understanding of the overall expectations in respect to the user interface, the various modes of operation, and the diversity found in throughout the application.

Prior to the discussion of each of the basic modes of operation, it is necessary to understand the basic method of graphical interface with the system. As figure 49 illustrates, the graphical environment is controlled through the use of the mouse. Lines are drawn simply

by pointing at the beginning location, pushing the mouse button, dragging the mouse to a new location, and again pushing the mouse button to complete the line segment. Other operations, such as primitive generation, extrusions, convergences, and geometric editing, require the mouse button to be depressed while moving about the graphics environment. This method of point, select, drag, and release, is prevalent throughout the modeler, and is utilized by every mode of operation.

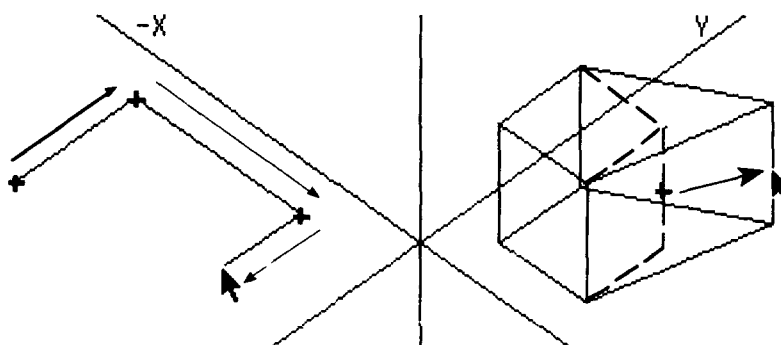


Figure 49: Basic user interface in the graphical environment.

From this point, the reader should be aware of the C•Mod environment, the menu command structure, the file structure, and the basic operational interface. Further discussion on these topics can be found in sections 3.1 and 3.2 of this chapter. The following section presents each mode in the order found on the menu. The icons presented are the ones found in the tool box after selection of the appropriate mode of operation. A brief discussion of the icons, as well as an example of the results produced are provided to illustrate the performance expected within each mode of operation.

3.3.1 Two-Dimensional Drawing

Basic line and polygon drawing can be performed in C•Mod under the 2-D mode of operation. In this mode, the tool box contains eight primitives (figure 50). By selecting any

one of these primitives, the user has the ability to generate two-dimensional objects. The 2-D mode is seldom used for most of the modeling process. However, it is required when creating objects of revolutions such as a torus or sphere.

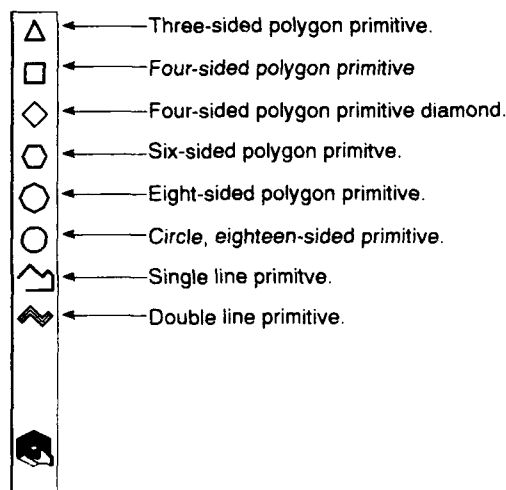


Figure 50: The tool box of icons in the two-dimensional drawing mode.

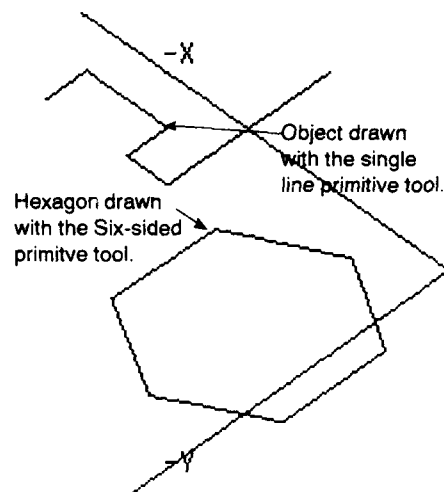


Figure 51: An example of drawing in the two-dimensional drawing mode.

3.3.2 Three-Dimensional Extrusions

The most fundamental mode of operation in the modeler is that of extrusion. Extrusions provide a quick and efficient method of generating three-dimensional entities within the modeler. Upon selecting the Extrude mode of operation, the system will display the available primitives in the tool box (figure 52). By selecting any of these primitives, the user has the ability to generate solid objects using the boundary-representation method of solid modeling. Figure 53 illustrates an example of two solid objects generated in this mode. Note the objects are surface rendered to provide a visual distinction of a solid element. With the show light source option disabled, the modeler will display the objects as wire-frame representations of the object.

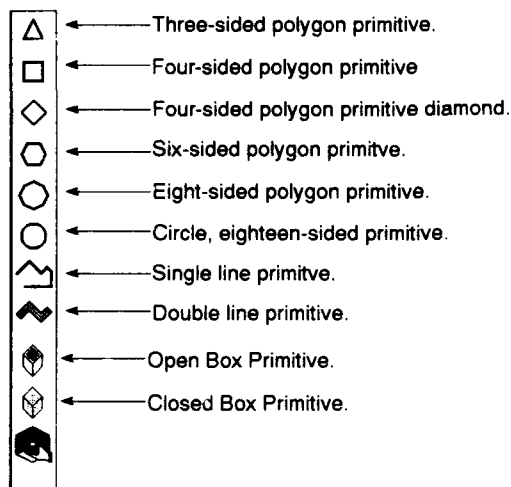


Figure 52: The tool box of icons in the Extrude mode.

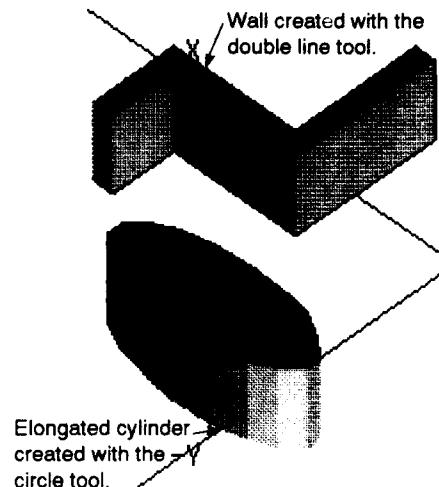


Figure 53: An example of creating solid objects in the extrude mode.

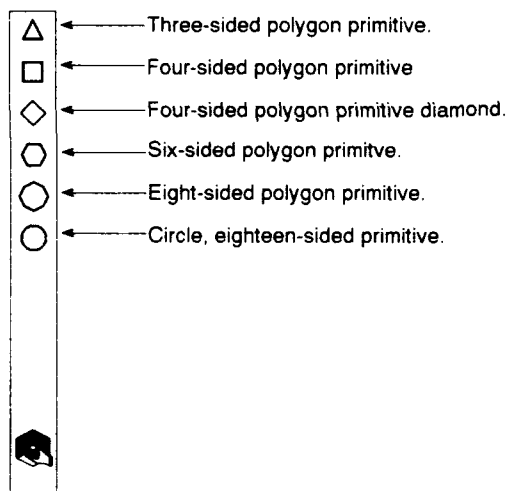


Figure 54: The tool box of icons in the converge mode.

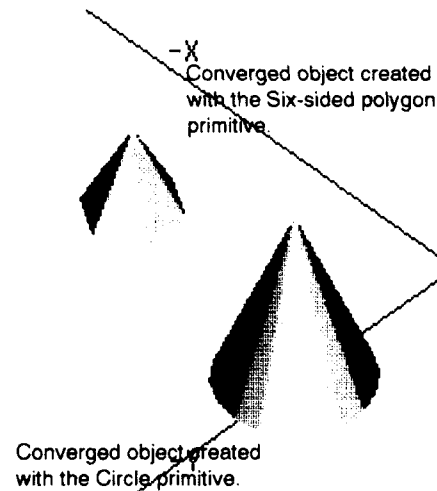


Figure 55: An example of creating solid objects in the converge mode.

3.3.3 Three-Dimensional Convergence

The converge mode is similar to the extrude mode in that it creates solid objects from a selection of primitives. However, in this mode of operation, the top face of the object is

created at an apex point which results in a pyramidal structure. Upon selection of the converge mode, the tool box will display the available primitives used to generate converged objects (figure 54). In a manner similar to the extrusion mode, the user can easily generate the pyramidal solid objects. Figure 55 illustrates an example of two solids generated through the converge mode of operation. Once again the objects have been surface rendered to illustrate solidity.

3.3.4 Three-Dimensional Objects of Revolution

Another method of generating solid objects is through the revolve mode of operation. This mode allows two-dimensional objects to be used as templates to create solid objects of revolution. Upon selection of the revolve mode, the tool box with the four options will be displayed (figure 56). In order for this operation to generate an object of revolution, a 2D primitive must be present in the graphic window. After selection of the 2D object and an axis of rotation, the modeler will generate the solid objects. Two examples are illustrated in figure 57, one of a torus, and one of a sphere.

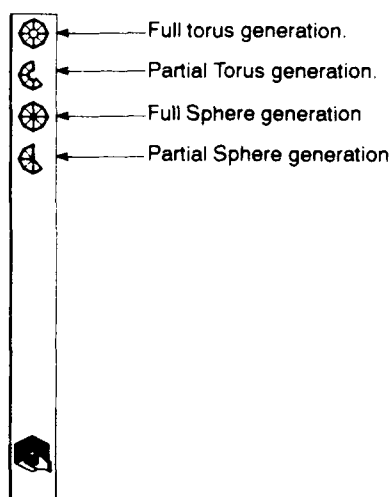


Figure 56: The tool box of icons in the revolve mode.

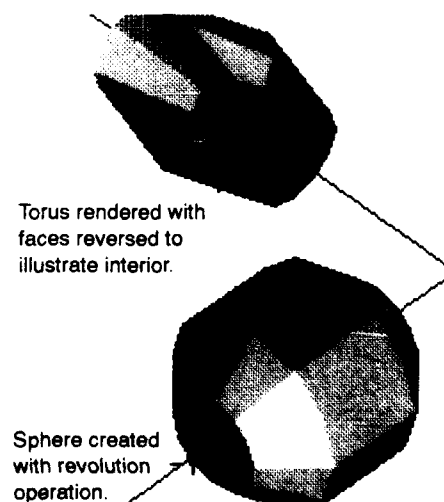


Figure 57: An example of creating solid objects in the revolve mode.

3.3.5 Geometric Editing

Geometric editing allows the modification of the geometry, or location, of any topological level, point, segment, face, volume, of an object. In this mode the user can translate, scale, rotate, and reform the object or parts of the object, along the active reference plane (figure 58). The results are displayed interactively as the mouse moves over the screen. This allows the object to "rubberband" or move as the mouse moves. Figure 59 illustrates a segment being translated along the x-y reference plane.

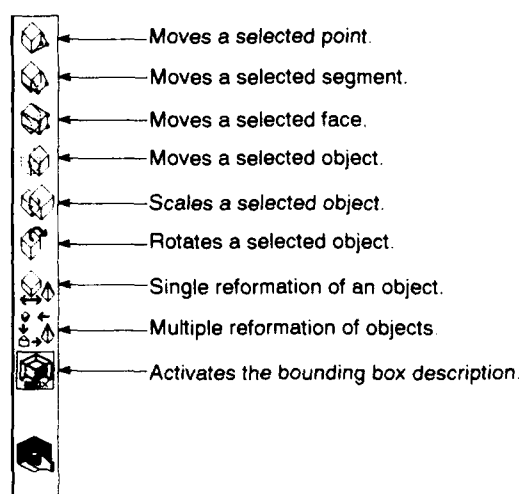


Figure 58: The tool box of icons in the g-edit mode.

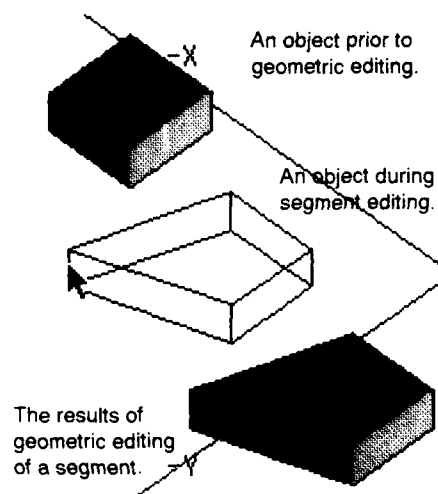


Figure 59: An example of editing solid objects in the g-edit mode.

3.3.6 Topological Editing

Topological editing allows the user to modify the physical structure, or topology, of an object. This type of editing enables the insertion or deletion of points and segments into the face or vertices of an existing object. By selecting the t-edit mode, the tool box will display a selection of operations which will perform these functions (figure 60). After selecting an

operation, the user selects the desired location for the insertion or element for deletion. In the event a valid selection was made, the system will perform the requested operation. Figure 61 illustrates an insertion of a segment into the face of an object. The segment has been translated to illustrate the existence on the face of the object.

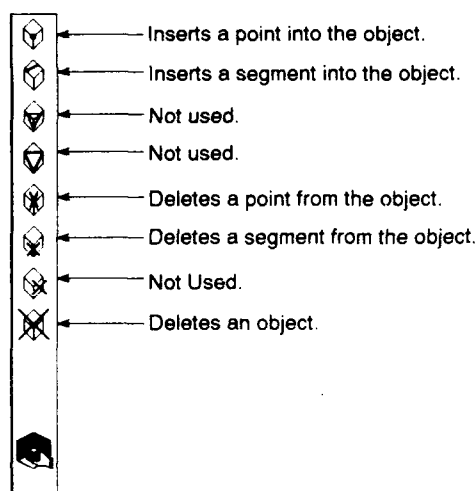


Figure 60: The tool box of icons in the t-edit mode.

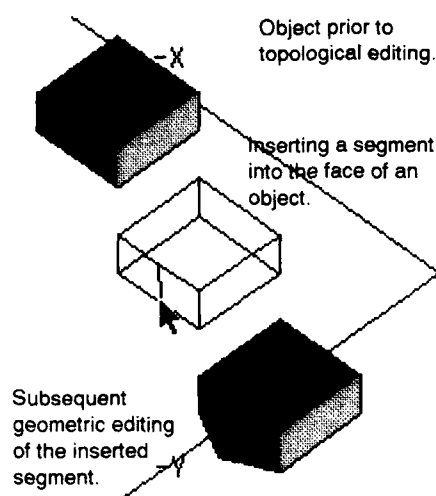


Figure 61: An example of editing solid objects in the t-edit mode.

3.3.7 Textures

Textures allows an enhanced visual representation of the solid objects by rendering visible surfaces with an architectural pattern. This form of surface mapping provides a few basic rendering capabilities to the solid modeler such as cross-hatching and brick patterns. The tool box (figure 62) illustrates the various patterns available to the user. A user can apply a texture pattern to the visible surfaces by selecting the desired pattern and then selecting the object to be rendered (figure 63). Since this is only a visualization technique, the patterns and renderings, are not stored with the object, and subsequent manipulation of the entity, or any operation which refreshes the image, will cause the pattern to be erased.

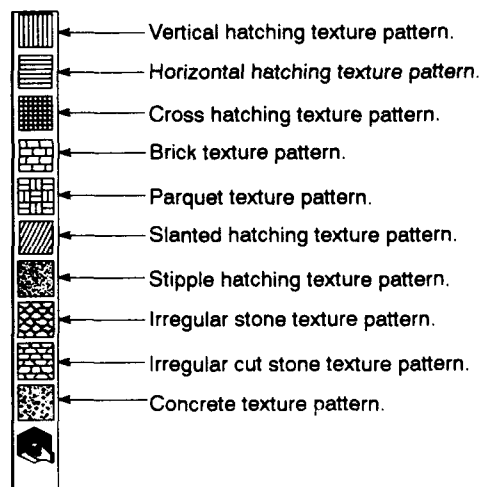


Figure 62: The tool box of icons in the textures mode.

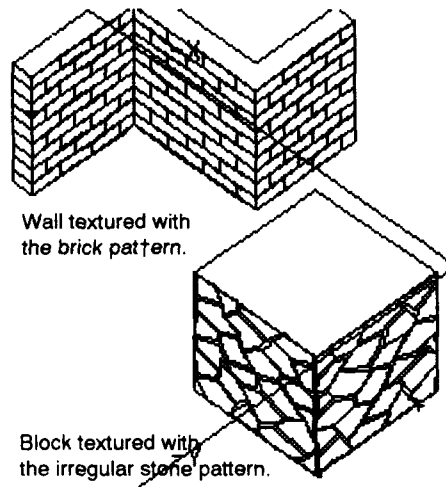


Figure 63: An example of rendering solid objects in the texture mode.

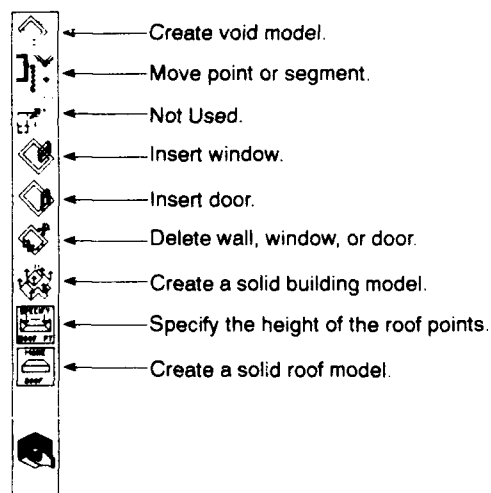


Figure 64: The tool box of icons in the void model mode.

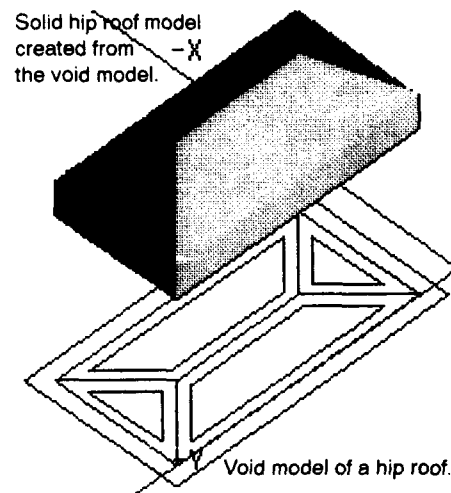


Figure 65: An example of generating a two-dimensional void model and a solid roof element from the void.

3.3.8 Void Modeling

The void modeling mode enables the user to generate and manipulate the axial skeletons of a two-dimensional void representation. A void model differs from a solid model in that it

defines an enclosed space. A wall, or enclosure is generated by deriving parallel lines to the axial skeleton. This representation allows the efficient generation of architectural plans which can be modified with the insertion or deletion of windows and doors as suited. Utilizing the void representation of a plan, a solid object, or building, can be generated, complete with the window and door openings. Additionally, the modeler has the capabilities to generate roof entities from the same two-dimensional void model. Figure 64 illustrates the tool box after selection of the void mode of operation. An example of the two-dimensional void model and a solid roof entity are illustrated in figure 65.

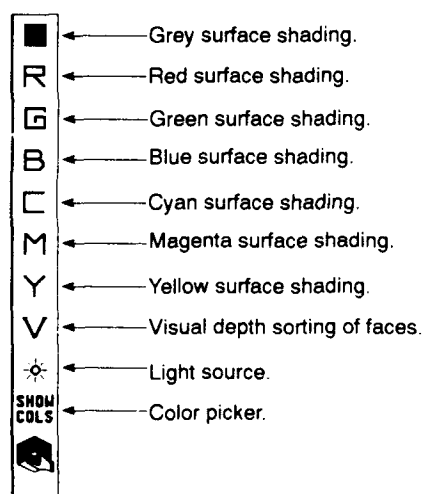


Figure 66: The tool box of icons in the colors mode.

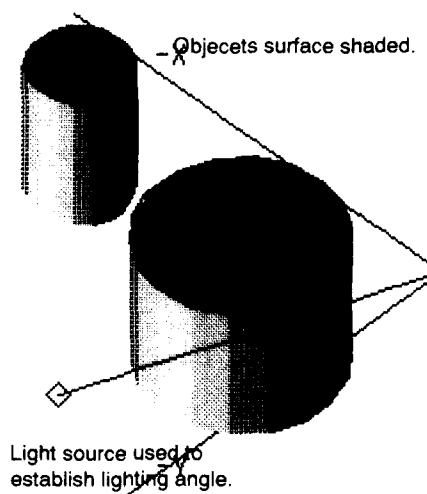


Figure 67: An example of rendering solid objects in the colors mode.

3.3.9 Color - Surface Shading

Surface shading provides the visualization of solidity. In the color mode, the user has the option of selecting and rendering the surfaces of objects. The colors available are indicated in the tool box and allow the assignment of different colors to different objects (figure 66). Unlike the texture option, colors are stored as part of the internal specifications of the object.

In the event a rendered scene is desired, the option show light source will render all objects with their appropriate color. An additional feature is the ability to change the direction and placement of the light source. This allows the surfaces of the objects to be rendered in various hues depending upon their angle with the light source.

3.3.10 Constraint-Based Modeling

The constraint mode of operations does not directly effect the modeling process itself. It does, however, effect how an object will behave while being subjected to the previous operations. This mode allows the user to specify design knowledge, or constraints, upon the generation and manipulation of the entity including its interaction with other entities. Since the constraint mode has several subordinate menu selections and operations, a complete description of the operations of constraint-based modeling are presented in section 3.4. To acquaint the reader with the mode however, the tool box of the various constraints which can be applied to the entity prior to, or after generation are illustrated in figure 68. In addition, figure 69 provides an example of interactive modeling within dimensional constraints.

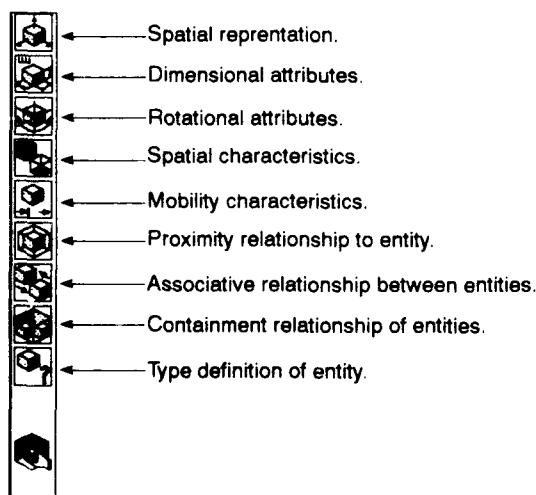


Figure 68: The tool box of icons in the constraints mode.

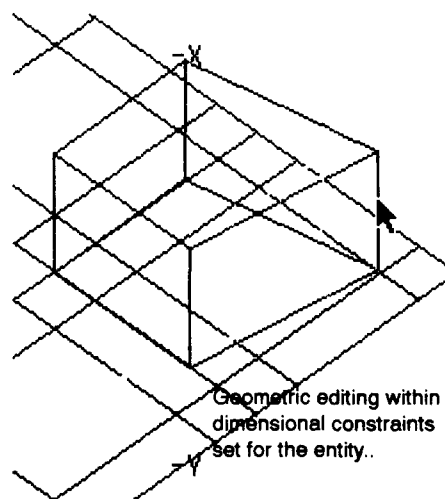


Figure 69: An example of editing solid objects (g-edit) with apply constraints active.

3.4 C•Mod Constraint Operations

C•Mod provides additional features which enable it to perform as a constraint-based solid modeler for architectural applications. This modeler, which can be utilized as a tool for Computer-Aided Architectural Design, allows for the definition and implementation of specific design knowledge which constrains the behavior of user definable three-dimensional entities.

The constraint-based solid modeler provides four main functional capabilities with which the user may apply to architectural applications. 1) The ability to specify the design knowledge applicable to user definable three-dimensional entities including entity, or physical characteristics, relational characteristics, and type identification. 2) The ability to modify, store, and retrieve the entity specifications provided through knowledge specification. 3) The ability to manipulate the three-dimensional entity in a manner which is consistent with the behavioral characteristics dictated by the entity specifications. 4) The ability to extract, query, information from an entity, or collection of entities, which has been provided by the specifications of that entity, as well as information derived from the manipulation and creation of the entity. This section elaborates on the constraint operations introduced in the previous section and includes a discussion of the sub-modes of the constraint operations complete with illustrations of the application to the modeling process.

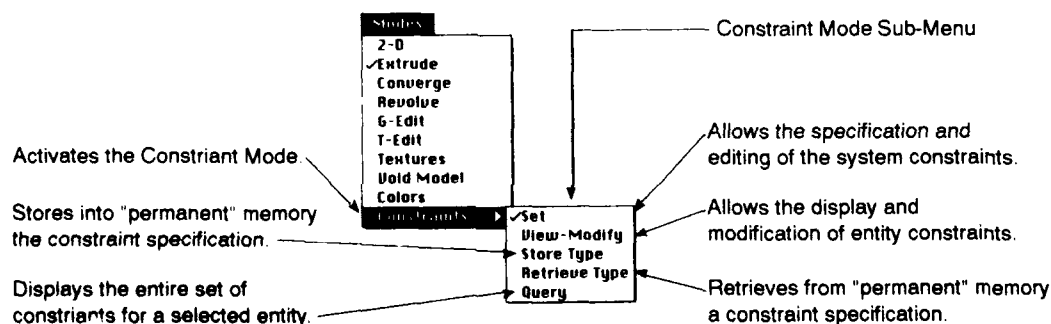











Figure 70: The constraint mode sub-menu.

The primary operations of C•Mod are found under the **Modes** menu and are invoked by selecting one of the five basic constraint operations which are provided as sub-menu items under the **Constraints** selection (figure 70). Depending upon the selection, the appropriate icons will appear at the left of the screen as previously shown in the tool box (figure 68). The first five icons (    ) in the tool box represent the entity characteristics, the next three (  ) represent the relational characteristics, while the last () is the identification attribute. By selecting any of the available icons, the user can activate the appropriate dialog for that type of constraint, and interactively establish the criteria for that constraint/attribute for any selected entity. Each of the nine icon selections is supported by this form of interaction, and provides the user with the ability to set, view, and modify any of the constraints for any of the entities which have been created.

3.4.1 Set and View-Modify

To apply design knowledge to the interactive modeling process, that knowledge must be specified and made available to the modeler. This constraint-based solid modeler provides the knowledge specification capabilities, and allows the user to input specific design knowledge applicable to a specific entity. This is accomplished by selecting the **Set** and **View-Modify** options from the constraint sub-menu. The distinction between the two is that **Set** applies to the system set of constraints used prior to the generation of an entity, while **View-Modify** applies to the entities set of constraints after it has been created.


Input of design knowledge is provided through a dialog between the user and the system. Design knowledge includes information and constraints specifically applicable to spatial and volumetric entities such as a design space, a room description, or a building description, and is the critical component in the interactive constraint-based modeling process. Three types of design knowledge are represented, entity characteristics, relational

characteristics, and identifying characteristics. The following is a discussion of each type of constraints and their specific fields.

3.4.1.1 Entity Characteristics

Entity characteristics are the specific physical attributes which the entity is required to satisfy during the generation and manipulation of the entity in the modeling process. There are five major physical attributes which constrain the modeling process, spatial representation, dimensional attributes, spatial characteristics, rotational attributes, and mobility characteristics. Each of the five entity characteristics, their dialog interface, and an example of the impact upon the modeling process, are illustrated.

3.4.1.1.1 Spatial Representation

 Spatial Representation indicates the entity's general behavioral characteristic by selecting a solid or spatial representation through the dialog box presented after selecting the spatial icon and clicking on the graphic window (figure 71). The distinction between the two representations lies in the ability to contain mass. Solid elements are just that - solid - they contain mass. Spatial elements do not contain mass, they contain space, or a spatial void. It is important to note that solid entities cannot contain any other entities, while spatial entities may contain solid entities. An illustration of the two types of representations are shown in figure 72. Note that the system default representation is a solid entity.

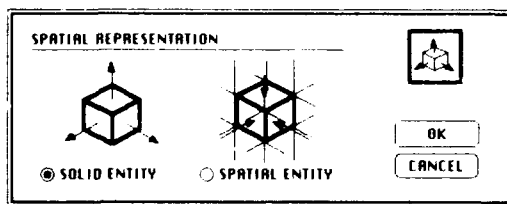


Figure 71: Dialog box for the spatial representation constraint.

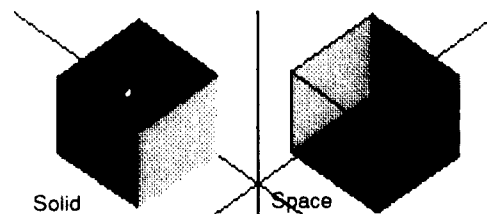


Figure 72: Illustration of the solid and space representation.

3.4.1.1.2 Dimensional Attributes



Dimensional attributes indicate a bounded box dimensional restriction upon the generation and manipulation of the entity (figure 73). The minimum, maximum, incremental and replication values may be set and activated for the width, height, and length attributes respectively. The minimum value establishes the smallest value that attribute may possess and acts as the initial set of values during generation. The maximum value establishes the largest value the entity may possess and acts as the replication threshold when the replication attribute is selected. An increment value establishes the incremental snap between the minimum values and the maximum values within the bounded box limitation. Replication will allow the successive regeneration of an additional entity when the maximum bounds have been exceeded. The system defaults for the dimensional characteristics are no minimum or maximum criteria, and no replication required.

As an aid in the visualization of the dimensional attributes, C•Mod provide a temporary set of construction lines (figure 74). These construction lines illustrate the dimensional criteria from the initial selection on the graphic window, and can be removed by regenerating the image on the screen.

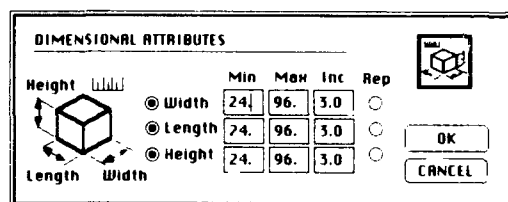


Figure 73: Dialog box for the dimensional characteristics constraint.

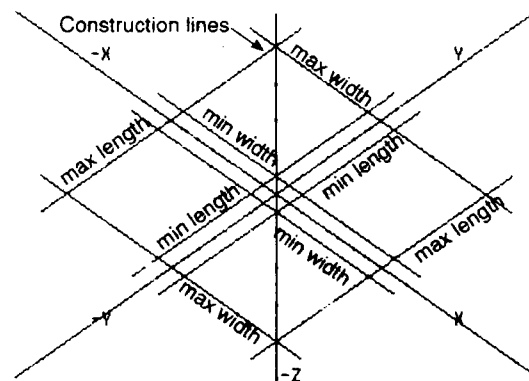


Figure 74: Illustration of the construction lines provided to visualize the constraints.

To illustrate the role in which the dimensional attributes perform in the modeling process, a series of illustrations are presented. C•Mod interactively constrains the generation (figure 75 and figure 76) and the manipulation (figure 77) of an entity by maintaining the dimensional characteristic of the entity in a bounding box (figure 78). This is required in order to maintain the dimensional data in three-space, and allows for the constraint to be applicable regardless of subsequent editing of the entity. As previously discussed, toggling the show bounding box option from the option menu will hide the bounding box from view.

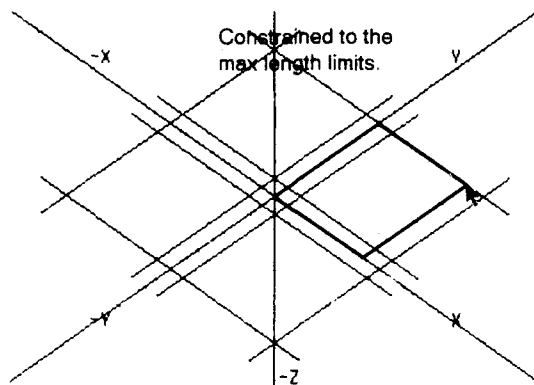


Figure 75: An illustration of the interactive generation of an entity within a dimensional set of constraints.

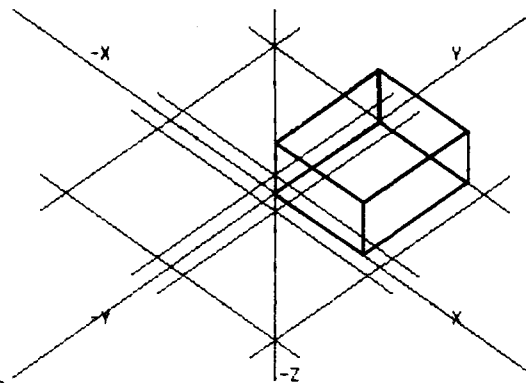


Figure 76: An illustration of an object created within the dimensional constraints.

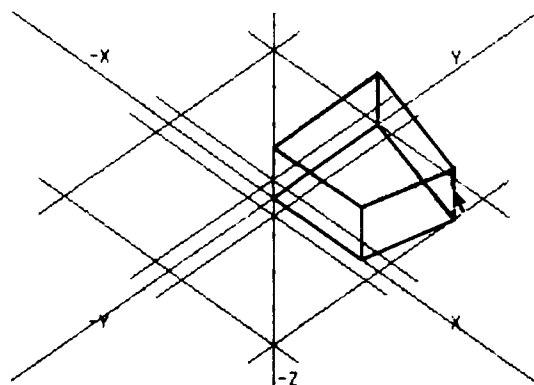


Figure 77: Geometric editing within the limitations of the dimensional constraints.

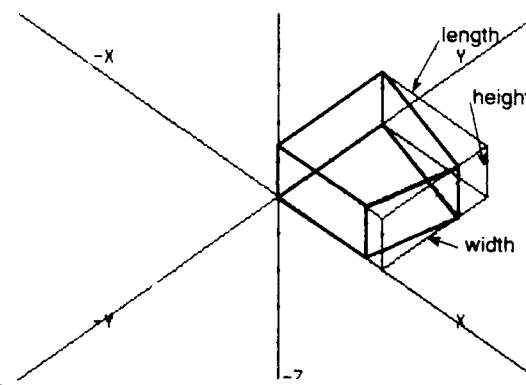


Figure 78: Visualization of the bounding box method of maintaining dimensional data.

3.4.1.1.3 Spatial Characteristics



Spatial characteristics provide in addition to, or in lieu of, dimensional attributes, constraints of area and volume which may be imposed on the entity. This allows the limitation of the generation and manipulation of the entity based upon minimum and maximum square and cubic units of measure (figure 79 and figure 80). The minimum value establishes the smallest area or volume that entity may possess. The maximum values establish the largest area or volume that entity may possess. An additional feature, orthogonality, constrains the entity to orthogonal continuity. The default values no area or spatial restrictions. This characteristic is not currently supported by C•Mod. The dialog box will appear, and design knowledge can be added to the constraint data structure, but the mechanisms to constrain the generation and manipulation have been considered part of the extensions to the modeler.

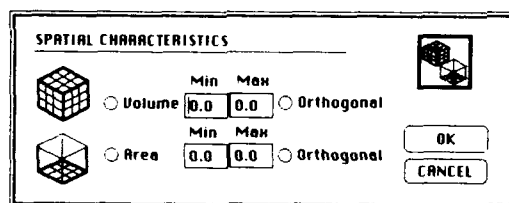


Figure 79: Dialog box for the spatial characteristics constraint.

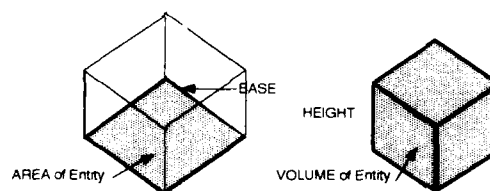


Figure 80: Illustration of the application of spatial characteristics to the object.

3.4.1.1.4 Rotational Attributes



Rotational attributes provide the means with which the entity can be constrained to a rotational level and degree of rotation during the geometric editing operations (figure 81). The level of rotation allows the selection and indication of allowable rotation in three-space, and includes activation and limitation of each of the three major axis (figure 82). The minimum and maximum values indicate the allowable range of rotation from 0 to 360 degrees in right-handed space. Additionally, the incremental degree of rotation about each axis can be

set. Default values for the rotational attributes are complete rotation about each axis with no restrictions. Although not implemented in this version of C•Mod, facilities for its extension have been provided.

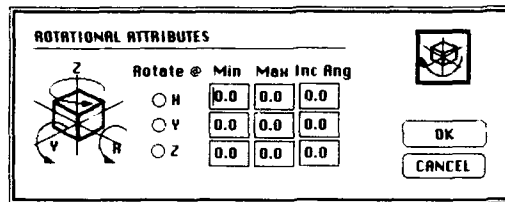


Figure 81: Dialog box for the rotational attributes constraint.

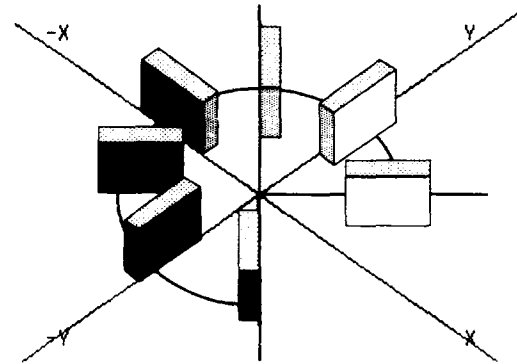


Figure 82: Illustration of the application of rotational attributes to the object.

3.4.1.1.5 Mobility Characteristics



The mobility characteristic establishes the degree of interaction the element will exhibit when acted upon by another element. Fixed mobility will not allow the element to react when interacted upon by another element, but remains fixed to its existing location. The reaction is translated to the interacting element. Free mobility allows the element to be freely acted upon by another element, and reaction is exhibited by the element itself. Upon selection of the mobility icon, and then selecting either the system or entity, C•Mod will display the mobility dialog box (figure 83). The user toggles between the two choices establishing the mobility characteristic for the selected entity. The default for this attribute is free mobility with no restrictions.

Since the mobility criteria fixes an object in three-space, the effects of this specification are visualized during the geometric editing of other entities. To illustrate this, two objects are

given opposite mobility characteristics (figure 84). The interactive manipulation of the free entity which causes it to interact with the fixed entity will not be allowed (figure 85). Conversely, translating the fixed object, (this is allowed since the mobility characteristic effects the behavior of other objects) in a manner which causes it to interact with the free object will result in the translation of the free object (figure 86). In effect, solids are moving solids, they cannot intersect when in the constraint mode.

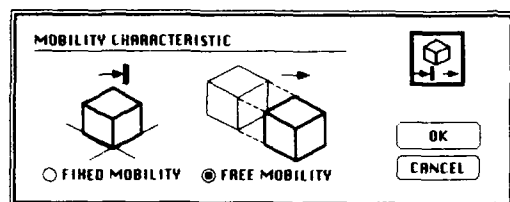


Figure 83: Dialog box for the mobility characteristics constraint.

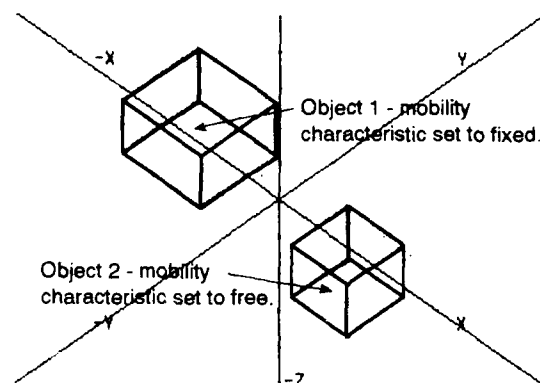


Figure 84: Assignments of the mobility characteristic of two solid entities.

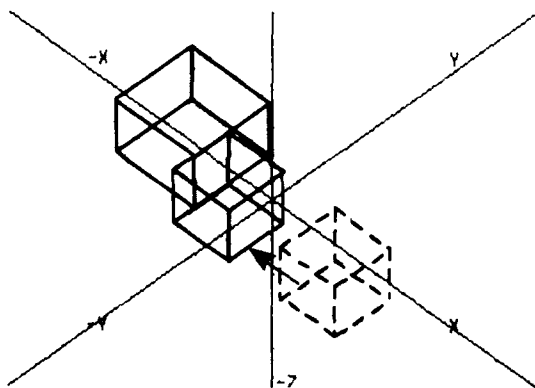


Figure 85: An illustration of the dynamic interaction between an object translating in three-space which confronts a fixed object. No further translation is allowed.

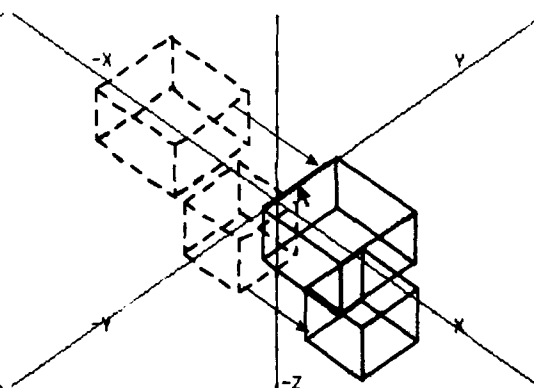


Figure 86: An illustration of the dynamic interaction between an object translating in three-space which confronts a free object. Both objects translate in the same direction.

3.4.1.2 Relational Characteristics

Relational characteristics are attributes the entity possess which indicate how that entity is to interact with other spatial entities. Three relational attributes, proximity relationships, containment relationships, and associative relationships, used separately, or in combination, constrain the manipulation of the entity within the three-dimensional modeling environment. The following is a discussion of each of the three relational characteristics, and provides an illustration of the effects upon the modeling process.

3.4.1.2.1 Proximity Relationships



Proximity relationships establish an offset zone around, for solid entities, and within, for spatial entities, the entity which provides a buffer, or clear zone, which cannot be encroached upon by any other object. A buffer zone is created by establishing an offset from the bounding box based on the width, length, and height distance from the entity. By utilizing an offset from the bounding box of the entity, the proximity data can be maintained in three-space regardless of the subsequent editing of the entity. The default is no proximity criteria specified. Upon selecting the proximity icon from the tool box, and the desired entity, the user can indicate the required offset in the width, length, and height dimensions. Figure 87 indicates the dialog box when a solid object is selected, and figure 89 indicates the dialog box for space entities. Note the difference between the two, a solid has an exterior proximity distance, while the space has an interior proximity distance.

To illustrate the effects on the modeling process, each case, solids and spaces, are graphically represented with and without a proximity zone established. Figure 88 illustrates how a proximity zone effects the relationship between solid entities. Figure 90 illustrates how the proximity zone effects the relationship between solids and spaces. The use of the proximity zone allows the user to establish a set-back, such as a zoning restriction, or offset, such as a ceiling to floor distance criteria, constraining the interactive modeling process.

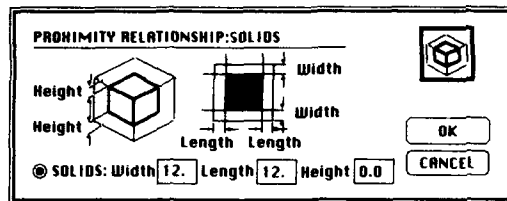


Figure 87: Dialog box for the proximity relationship for solid entities.

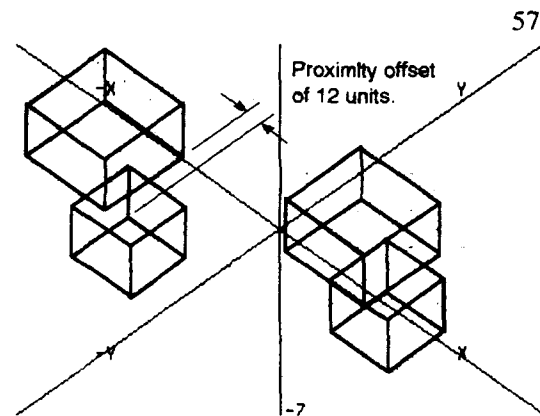


Figure 88: The interaction between entities with and without proximity relationships.

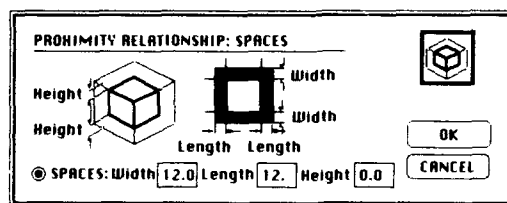


Figure 89: Dialog box for the proximity relationship for space entities.

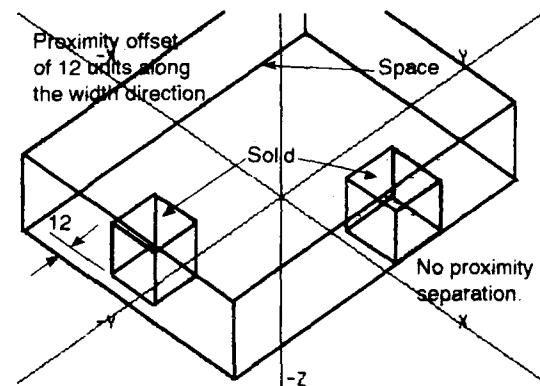


Figure 90: The interaction between a space entity and two solid entities. The space has a proximity offset established in the width direction only.

3.4.1.2.2 Containment Relationships



Containment relationships establish the relationship between dissimilar spatial representations. Depending upon the spatial representation given to an object, the available options will be either no containment and solid contained by a specified volume, or no containment and a spatial entity which contains user specified solid entities. No

containment allows the entity to be non-restricted in any space. If containment is desired, the entity can belong to a spatial entity, and thus contained by that entity, or contain solid entities, depending on the spatial representation. The default, regardless of the spatial representation, is no containment.

Upon selection of the containment icon, the system will prompt the user for information through a dialog box. Figures 91 and 92 illustrate the difference between the containment dialog for a solid and a space. The implementation of the containment relationship has been left as an extension to the C•Mod application. Therefore, the dialog boxes have no applicable use at the current time. It is important to note that spaces currently have the ability to contain solid entities, and will restrict the generation and editing of the entity to the bounds established for the space entity, as illustrated in figure 93.

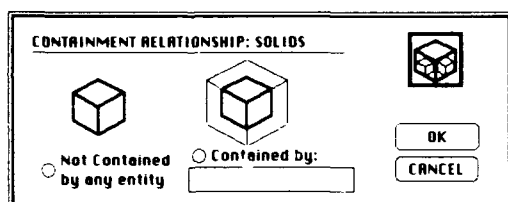


Figure 91: Dialog box for the containment relationship for solid entities.

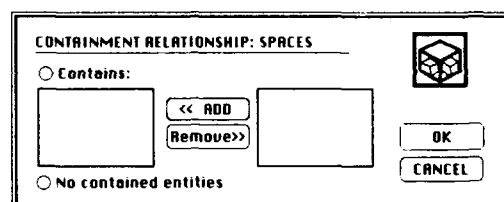


Figure 92: Dialog box for the containment relationship for space entities.

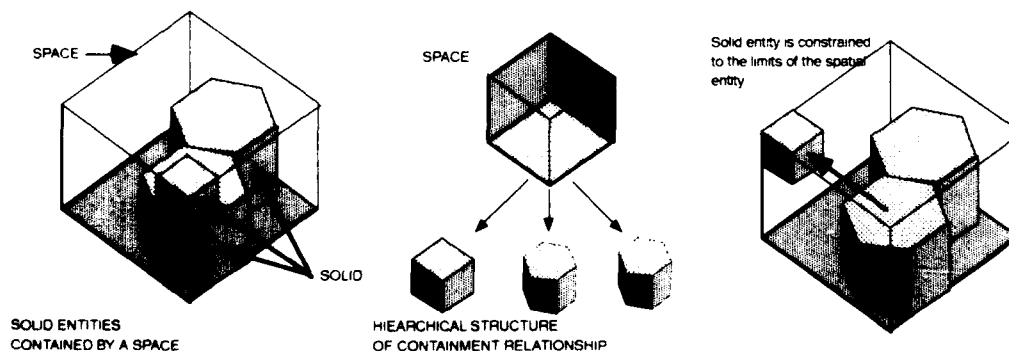


Figure 93: An Illustration of the containment constraint on the modeling process.

3.4.1.2.3 Associative Relationship



Associative relationships establish the response an entity will exhibit during the interaction with another entity. Unlike the mobility characteristic which effects the physical interaction between entities, the associative relationship effects the spatial relationship between interacting entities. Utilizing an offset from the proximity zones for each entity, the interaction between two or more entities will be affected by their associative relationship indicated, and respond in one of three actions, none, attract, or repel (figure 95). No relationship indicates that there is no inherent association required for that entity. Attracting relationship indicates that the object will attract, or try to attach itself, to another object which approaches the tolerance offset. Repel relationship indicates that the object will oppose, or repel another entity which approaches the tolerance offset. Upon selection of the association icon, the system will prompt the user to make a selection between the three options, and then indicate the type of entity which the association is to be applied (figure 94). The selected association will be applied to the interaction between the entity and all entities of the selected type.

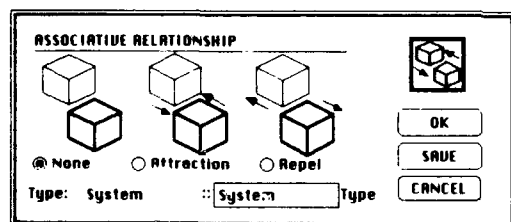


Figure 94: Dialog box for the associative relationship between entities.

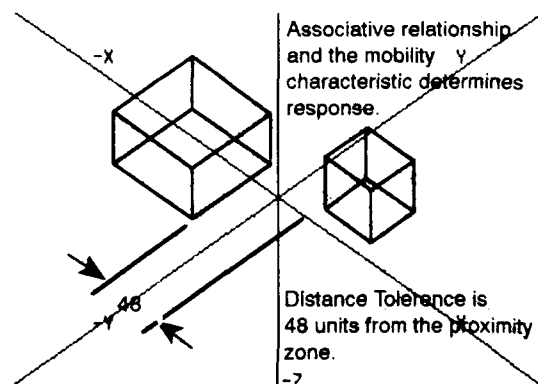


Figure 95: The offset from the proximity zone which determines activation of the associative relationship.

To illustrate the interaction between entities influenced by the associative relationship, several examples of attract and repel have been provided. The illustrations provide a point of reference which indicates the modeling scene prior to a translation and indicates the anticipated results based on the association and mobility of the entities in the scene. Following each of these reference views are the actual results of the translation in C•Mod. Since this is still view of an interactive and dynamic process, outlined representations of the object have been provided to indicate their position prior to translation.

Figure 96 illustrates the view of the modeling scene prior to a translation of an entity (object A) which will be translated within the offset tolerance of another entity (object B). The anticipated results are indicated by the direction vectors emanating from each of the entities. Both entities have an association with one another of attract. In addition, object B has a mobility characteristic set to free. As the user interactively translates the entity (object A) and approaches the second entity, object B will snap, or move in one continuous motion, to the face of the translating entity (figure 97). As object A continues to translate, object B will remain attached. To free the association between the two entities, a quick and continuous movement of the original entity will release the attraction.

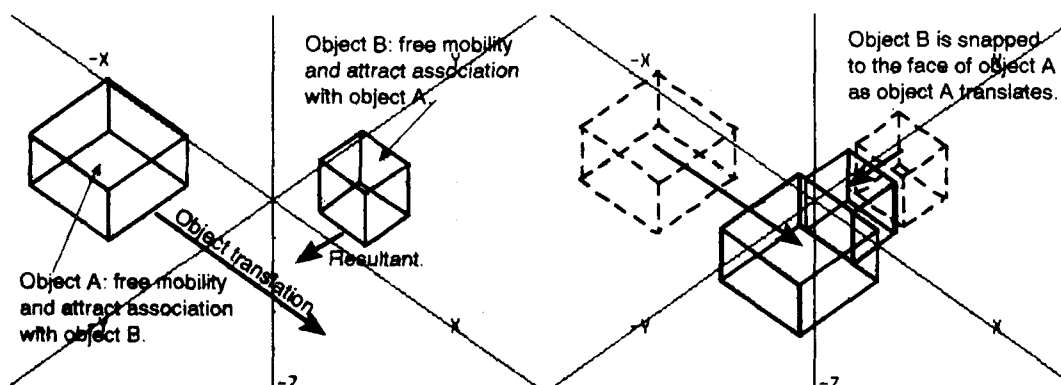


Figure 96: The anticipated translation and resultant interaction involving an entity with free mobility and an attract assoc.

Figure 97: The resulting interaction between an entity with free mobility and an attract association.

With the mobility of the second object set to fixed, the anticipated results indicate a different reaction. Figure 98 illustrates the view of the modeling scene prior to a translation of an entity (object A) which will be translated within the offset tolerance of another entity (object B). The anticipated results are indicated by the direction vectors emanating from each of the entities. Both entities have an association with one another of attract. As the user interactively translates the entity (object A) and approaches the second entity, object A will snap, or move in one continuous motion, to the face of the fixed entity, object B (figure 99). Object A now can only translate along the face of the fixed entity and cannot extend beyond the bounds of this constraint. As with the previous example, to free the association between the two entities, a quick and continuous movement of the original entity will release the attraction and permit free movement in three-space without restrictions.

In either case, attract with free mobility, or attract with fixed mobility, the interacting objects will attempt to attach themselves to one another. This characteristic is useful when considering the placement of solids within a space, such as kitchen cabinetry along a static wall. The wall can be set to fixed, and the association between it and the cabinets can be set to attract, thus ensuring that the cabinets are placed against a wall.

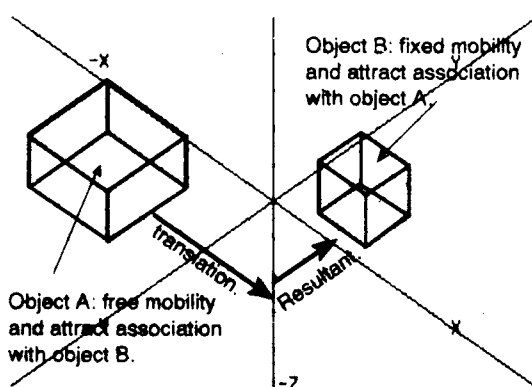


Figure 98: The anticipated translation and resultant interaction involving an entity with fixed mobility and an attract assoc.

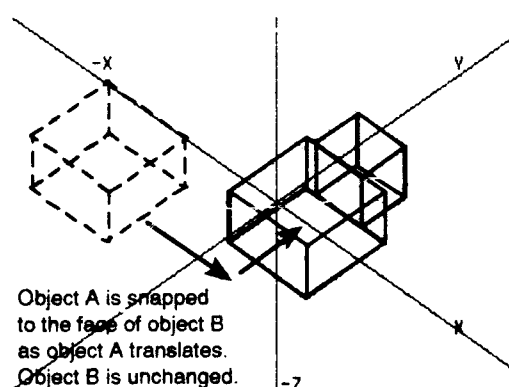


Figure 99: The resulting interaction between an entity with fixed mobility and an attract association.

The modeling effects of the repel association follow in similar fashion to that of the attract association. The difference is in the reaction of the entities. Entities with an repel association will attempt to reject the attachment or encroachment upon the space surrounding the entity. They will behave as magnets with similar charges attempting to repel one another. An illustration of the repel association with both free and fixed mobility is presented to illustrate the visualization of the effects on the modeling process.

Figure 100 illustrates the view of the modeling scene prior to a translation of an entity (object A) which will be translated within the offset tolerance of another entity (object B). The anticipated results are indicated by the direction vectors emanating from each of the entities. Both entities have an association with one another of repel. Object B has a mobility characteristic set to free. As the user interactively translates the entity (object A) and approaches the second entity, object B will reject the advance of object A by moving out of its path of travel (figure 101). As object A continues to translate, within the tolerance zone of object B, object B will continuously move away from object A. This association will continue as long as object A encroaches upon this tolerance zone. A movement of object A in a direction away from object B does not effect object B.

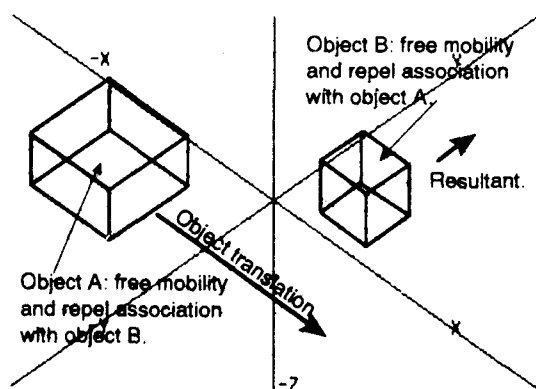


Figure 100: The anticipated translation and resultant interaction involving an entity with free mobility and a repel association.

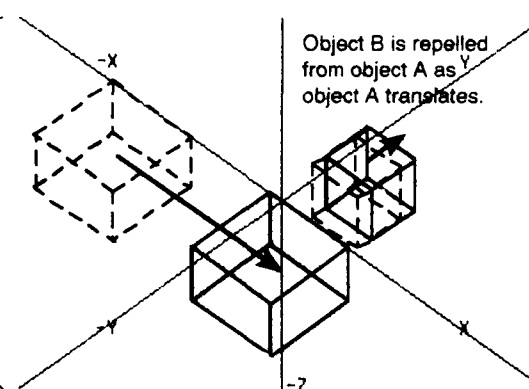


Figure 101: The resulting interaction between an entity with free mobility and a repel association.

With the mobility of the second object set to fixed, the anticipated results indicate an opposite reaction. Figure 102 illustrates the view of the modeling scene prior to a translation of an entity (object A) which will be translated within the offset tolerance of another entity (object B). The anticipated results are indicated by the direction vectors emanating from each of the entities. Both entities have an association with one another of repel. As the user interactively translates the entity (object A) and approaches the second entity, object A will be rejected, or repelled away from object B (figure 103). As object A continues to translate, within the tolerance zone of object B, object B will continuously reject object A and force it back from its tolerance zone. This association will continue as long as object A encroaches upon this tolerance zone.

The use of the association relationship provides a method of realistically modeling the behavior of interacting entities. When used in combination with the other constraint characteristics and relationships, the modeling environment becomes responsive to the physical and relational quality of solid and space entities. In an interactive modeling environment, this translates to a dynamic and realistic modeling of simulated real world entities which will behave in accordance with anticipated and results.

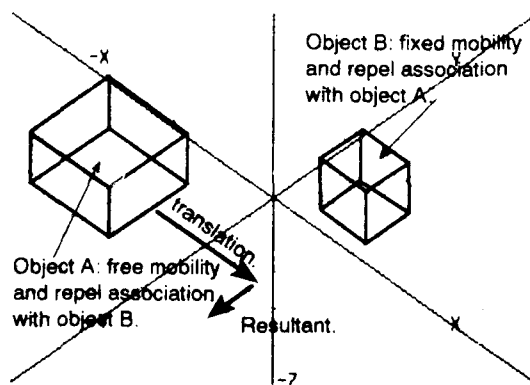


Figure 102: The anticipated translation and resultant interaction involving an entity with fixed mobility and a repel association.

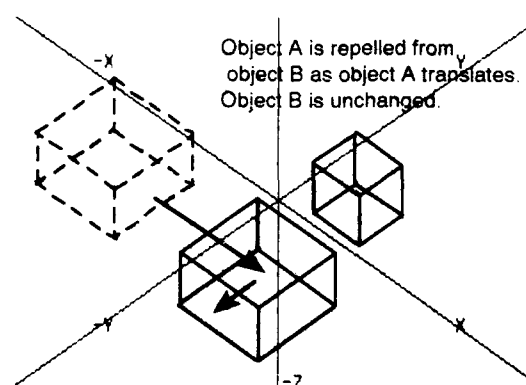



Figure 103: The resulting interaction between an entity with fixed mobility and a repel association.

3.4.1.3 Identification Attribute

The identification attribute allows the unique identification of a set of constraints which have been established for a given entity. This allows the system to distinguish between differing types of entities when establishing associations, and provides a means of identifying a set of constraints during the I/O operations. Note that the type definition attribute is required when storing and retrieving constraint information.

3.4.1.3.1 Type Definition

 Type Definition is a unique identifiable descriptive attribute associated to the set of constraints representing an entity. The specification of a type definition to a specific set of constraints provides the capability to store and retrieve the entire set of constraints with a single identifying macro. By allowing the set of constraints to be identified as a single type definition, the set of constraints can be created and stored once, and then retrieved and assigned to entities with a single reference indicating that type. This ability to assign a complete set of constraints with a single reference allows quick and efficient use of the design knowledge about entity types without specific knowledge about all the particular attributes required to represent that type. A type definition is required when storing a set of constraints into memory. Upon selecting the type definition icon, the system will prompt the user for a name of the entity (figure 104). The default name is the current name assigned to the system specifications.

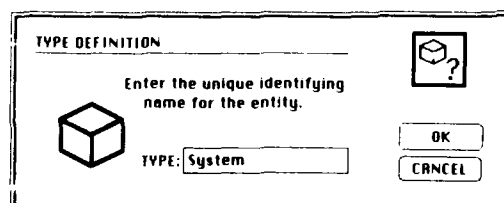


Figure 104: The type definition dialog box.

3.4.2 Store Type and Retrieve Type

C•Mod provides the user to store and retrieve specific sets of design knowledge as an entire type. This allows the specification of an entire class of objects to be stored for use later without respecification. C•Mod distinguishes this file structure independently from the project files specified in section 3.1.1 and 3.2.2. Figure 105 illustrates the constraint type definition icon used to represent this information. The standard file dialogs presented in the following section will only access these types of files if they are present. To facilitate this specific form of file processing, C•Mod provides two specific I/O operations under the constraint sub-menu, **Store Type** and **Retrieve Type**. The following section will elaborate on the use of each of these operations.



Figure 105: The type definition file.

3.4.2.1 Store Type

Store Type allows the storage of the complete set of constraint specifications based on the entity type definition to "permanent" memory, thus retaining the design specifications for use at a later point in time. Upon selection of the **Store Type** sub-mode, and subsequent click on the graphic window, the system will display a standard file dialog box which will prompt the user for a type definition in the event one has not been provided, and will write the type definition to a file in "permanent" memory. In the event that a type definition previously exists, the user will be prompted to change or accept the name of the definition for storage. Figure 106 illustrates the standard file dialog for saving constraint files.

3.4.2.2 Retrieve Type

Retrieve type allows the retrieval of a complete set of constraint specifications based on the type definition which have been stored in "permanent" memory. Upon selection of the **Retrieve Type** sub-mode, and subsequent click on the graphic window, the system will display a standard file dialog box which will prompt the user for the identification of the type definition which is to be retrieved, and will attempt to access that set of constraint specifications. In the event that the constraints have been stored with the store type operation listed above, the system will then proceed to read the file into the system specifications for use by the system. Figure 107 illustrates the standard file dialog box for file retrieval. Any new entities created under this set of system specifications will contain and exhibit the constraints of the retrieved type.

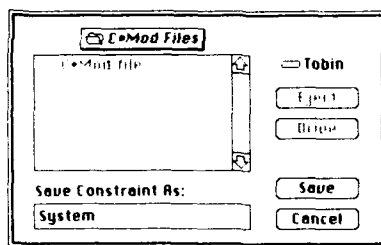


Figure 106: Store file dialog box.

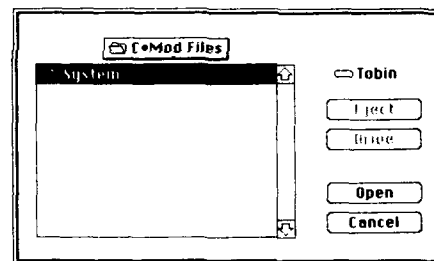


Figure 107: Retrieve file dialog box.


3.4.3 Query

As an alternative to viewing the individual constraints upon a selected entity, C•Mod provides the user with the ability to query the systems to display the complete set of design knowledge pertaining to a specific entity, or to the system specifications. By selecting the **Query** option under the constraints sub-menu, the user can retrieve the entire set of constraints which have been specified for the entity or for the system. Subsequent clicking, or selection of

the entity will display the query dialog box illustrating all of the constraints set for that particular object (figure 108). In the event that the constraints for the system are desired, any selection on the graphic window which is not an object will display the same dialog box for the system specifications. This method allows the immediate and efficient visualization of the entire set of constraints from a single command.

The following constraints have been set:

Entity Name: System
Entity Type: Solid Entity
Mobility: Free



Dimensional:	min	max	inc	active	replicate
width:	24.0	96.0	3.0	Yes	No
length:	24.0	96.0	3.0	Yes	No
height:	24.0	96.0	3.0	Yes	No

Spatial:	min	max	active	ortho
volume:	0.0	0.0	No	No
area:	0.0	0.0	No	No

Rotational:	max	min	inc	active
xrot:	0.0	0.0	0.0	No
yrot:	0.0	0.0	0.0	No
zrot:	0.0	0.0	0.0	No

Proximity:	width	length	height	active
Solid :	12.0	12.0	0.0	Yes

Figure 108: The query dialog box.

CHAPTER IV

INTERNAL WORKINGS AND ALGORITHMS

The introduction of design knowledge to the interactive solid modeling process requires that the modeler provide an efficient and economic method of determining and evaluating the criteria, or constraints. This is a result of the large number of calculations in a three-dimensional environment. Each additional entity, as well as each additional constraint, or representation of design knowledge, further compounds the interactive modeling process. This chapter discusses the internal composition of the constraint-based solid modeler, C•Mod, and presents the main data structures used to represent solid objects and design knowledge. Additional discussion will focus on the algorithmic outline of each of the main procedures used to provide the association and interaction between solids and constraints.

4.1 Discussion of the Internal Composition

To achieve the goals and objectives stated in chapter I, the main capabilities of the constraint-based solid modeler must allow, and provide for, the interactive use of design knowledge to constrain the generation and editing functions of a modeler. To accomplish this interaction, the constraint-based solid modeler must represent two types of data, object and constraint. The object data structure represents the entity itself, and is the primary data structure for the solid modeler. Section 4.1.1 discusses the object data structure in detail, and illustrates the internal representation within the modeler. The second primary data structure for the constraint-based solid modeler is the constraint data structure. This data structure represents the design knowledge about the entity specified in the object data structure. Section

4.1.2 discusses the constraint data structure in detail, and illustrates the internal representation with the modeler.

4.1.1 Object Data Structure

The entity in the constraint-based solid modeler is represented internally in an object data structure. This internal representation models the solid object by utilizing a boundary representation. A boundary representation contains descriptive data at six topological levels, object, volume, face, curve, segment, and points. Each topological level is internally represented as a separate array with references between the different levels. Figure 109 illustrates a hierarchical diagram indicating the topological levels from the most general to the most specific.

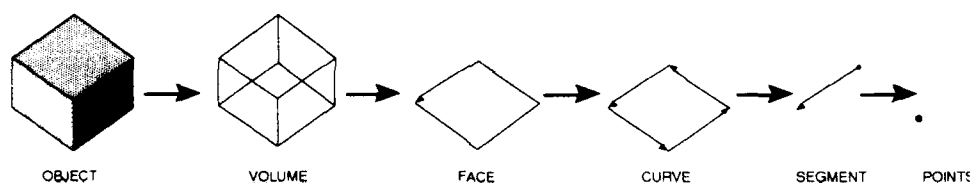


Figure 109: Topological levels of an object.

Boundary representations follow a hierarchical description indicated by the topological levels of the object. An **object** is defined by the composition of a volume, and contains attribute information such as color, and constraints. A **volume** is defined as a collection of faces which represent each facet, or face, of an entity. Volumes contain a list of all of the faces which represent the surface of the object. To represent each surface, **faces**, provide a means of storing a collection of curves, each representing a continuous edge found along the planar surface. Thus, a face with an opening is represented by two curves, one for the outer edge, or boundary, of the surface, and one for the outer edge, or boundary, of the opening. These

curves are a collection of continuous segments which define the boundary of the curve. Curves contain a list of all of the segments which construct the curve, along with the number of points contained within each curve. **Segments** connect points, and allow for the tracing of each edge along the curve. At the lowest topological level, **points** represent the actual Cartesian coordinate values which define the object. By constructing the solid object in this manner, the points which define the entire object are stored only once, regardless of the number of times it is used by adjacent faces. Figure 110 illustrates the internal object data structure. In addition, a complete description of each component is presented to fully illustrate the composition of the object.

OBJECT DATA STRUCTURE :

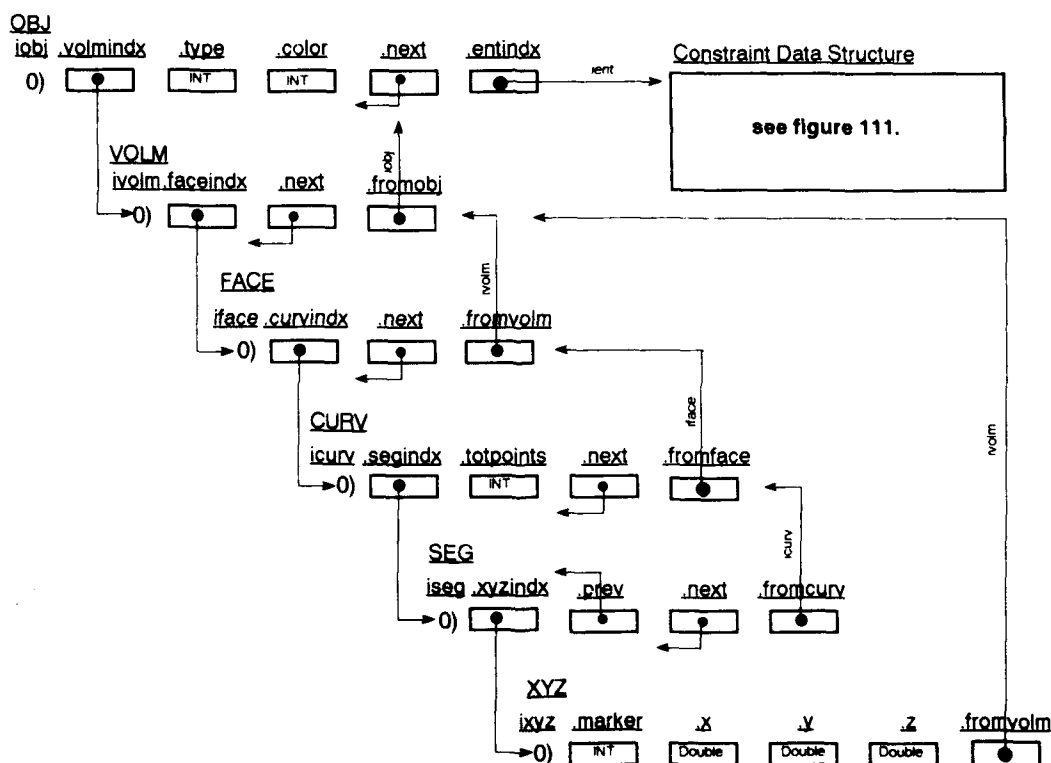


Figure 110: Diagram of the object data structure.

Obj, which is at the highest topological level internally, represent the object as a collection of all other topological levels, and include five elements, type, volmindx, color, next, and entindx. The following describes each of these elements:

- **type**: a description of the type of object generation (extrusion, convergence).
- **volmindx**: an index to the first volume of the object.
- **color**: a color descriptor which indicates the color of the object.
- **next**: an index to the next object in the object array structure.
- **entindx**: an index which is the link between the object data structure and the constraint data structure, and allows the entity to possess design knowledge.

Volm is the next topological level and represent the collection of volume entities which are formed to compose the object. Typically, a collection of volumes create an object. However, for this modeler, an object and volume are similar in nature. A volume structure is composed of three elements, fromobj, faceindx, and next. The following describes each of these elements:

- **fromobj**: an index to the face array, and indicates the first face among the list of faces which make up the object.
- **fromobj**: an index which serves as a reference to the object which the volume belongs. This is the link between objects and volumes.
- **next**: an index to the next volume in the volume array structure.

Face is an array which indicates each face of a volume and is made up of a collection of curves comprising each face of an entity. The face array is composed of three elements, fromvolm, curvindx, and next. The following describes each of these elements:

- **fromvolm**: an index to the volume which is the owner of this faces.

- **curvindx:** an index to the curve array indicating the list of curves which make up each face.
- **next:** an index to the next face in the face array structure.

Curv is an array which contains a list of all curves for each face of the object. Curves include the outer curve of a solid entity as well as any interior curve representing opening within the face of the entity. The curve structure contains four elements, fromface, segindx, totpoints, and next. The following describes each of these elements:

- **fromface:** an index to the face which owns the curve, indicating which face this curve belongs to.
- **segindx:** an index to the first segment of the curve, indicating the beginning of the curve at the segment level.
- **totpoints:** indicates the total number of points or segments which make up the curve, this acts as a loop counter when tracing the curve.
- **next:** an index to the next curve in the curve array structure which belongs to this particular face.

Seg is an array which contains a two-way linked list of all segments which make up the each curve in the face. Segments are a connection between two points in the data structure are linked to the previous and next segments of the curve. The segment structure contains four elements, fromcurv, xyzindx, prev, and next. The following describes each of these elements:

- **fromcurv:** an index to the curve which owns this segment, indicating which curve this segment belongs to.
- **xyzindx:** an index to the points (xyz) array, indicating the beginning point of the segment.

- **prev:** an index to the seg array, indicating the previous segment in the list of segments which compose the curve.
- **next:** an index to the seg array, indicating the next segment in the list of segments which compose the curve.

XYZ is an array which contains the lowest topological level of information required to represent the solid object, and stores the coordinate values of the points in three-space. The xyz structure contains five elements, fromvolm, marker, x, y, and z. The following describes each of these elements:

- **fromvolm:** an index to the volm array, indicating the volume which owns this set of points.
- **marker:** a open element which serves as a flag for various operations within the modeler.
- **x:** a double value indicating the Cartesian coordinate value of the point in the x direction.
- **y:** a double value indicating the Cartesian coordinate value of the point in the y direction.
- **z:** a double value indicating the Cartesian coordinate value of the point in the z direction.

4.1.2 Constraint Data Structure

Design knowledge is represented internally by the constraint data structure. The constraint data structure is composed of two information levels, and five constraint tables. The two information levels are comprised of the entity level which stores specific design knowledge input by the user, and the characteristic and relationship level, which is determined by the system. The importance of these elements lies in the access to the design data which is

stored in the constraint tables. Design data, when interpreted interactively through the user, is stored in one of five tables, data, association, containment, bounding box, proximity box, and bounding sphere. Figure 111 illustrates the constraint data structure and its information flow.

CONSTRAINT DATA STRUCTURE FOR AN ENTITY:

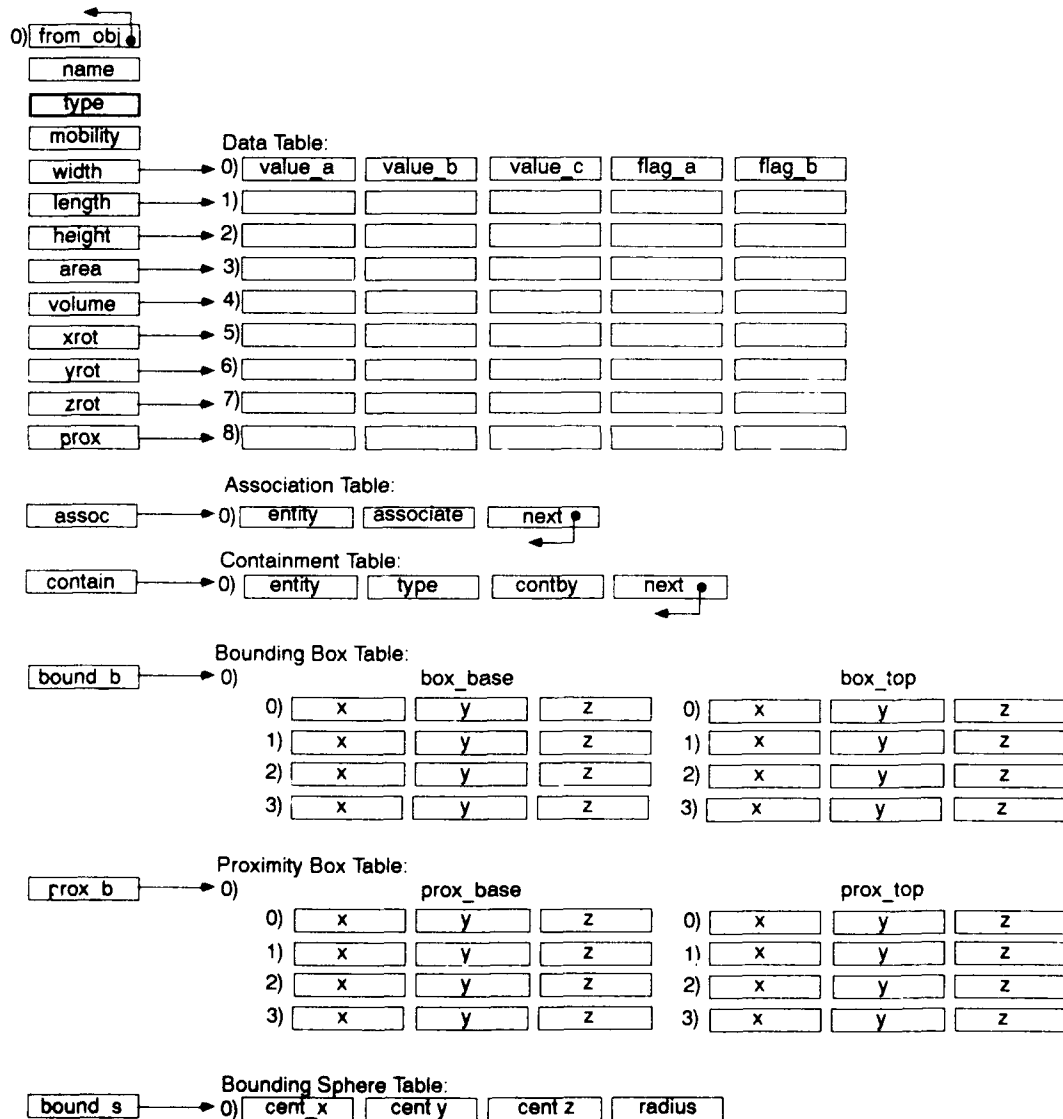


Figure 111: Diagram of the constraint data structure.

Entity, which is at the highest level internally, is an array of the critical information required by the constraint modeler, and is indexed into through the object's `entindx` field. The entity field is composed of eighteen elements: `fromobj`, `name`, `type`, `mobility`, `width`, `length`, `height`, `area`, `volume`, `xrot`, `yrot`, `zrot`, `proximity`, `assoc`, `contain`, `bound_b`, `prox_b`, and `bound_s`. The following is a discussion of each of these elements:

- **fromobj**: an index to the object data structure, indicating the object which owns this particular set of constraints or design data.
- **name**: The name represents a specific identity for a collection of design data or constraints, and is used to store and retrieve the data as that collection.
- **type**: indicates the entity's spatial representation either as a solid or as a spatial. A 0 (zero) indicates a space, and a value of 1 (one) indicates a solid object.
- **mobility**: a specific value indicating the entities ability to react to the interaction between other entities. A 0 (zero) value indicates fixed mobility, and a 1 (one) value indicates free mobility.
- **width**: an index to the data table, indicating the width attributes required for the entity. The values represented in the data table refer to the minimum width, maximum width, increment, active/inactive flag, and the replication flag, respectively. Negative values indicate that attribute is not applicable.
- **length**: an index to the data table, indicating the length attributes required for the entity. The values represented in the data table refer to the minimum length, maximum length, increment, active/inactive flag, and the replication flag, respectively. Negative values indicate that attribute is not applicable.
- **height**: an index to the data table, indicating the height attributes required for the entity. The values represented in the data table refer to the minimum height, maximum height, increment, active/inactive flag, and the replication flag,

respectively. Negative values indicate that attribute is not applicable. Negative values indicate that attribute is not applicable.

- **area:** an index to the data table, indicating the area attributes required for the entity. The values represented in the data table refer to the minimum area , maximum area, a null value, active/inactive flag, and the continuity flag, respectively. Negative values indicate that attribute is not applicable.
- **volume:** an index to the data table, indicating the volume attributes required for the entity. The values represented in the data table refer to the minimum volume, maximum volume, a null value, active/inactive flag, and the continuity flag, respectively.
- **xrot:** an index to the data table, indicating the xrot attributes required for the entity. The values represented in the data table refer to the minimum rotation angle, maximum rotation angle, an incremental angle, an active/inactive flag, and a null value, respectively.
- **yrot:** an index to the data table, indicating the yrot attributes required for the entity. The values represented in the data table refer to the minimum rotation angle, maximum rotation angle, an incremental angle, an active/inactive flag, and a null value, respectively.
- **zrot:** an index to the data table, indicating the zrot attributes required for the entity. The values represented in the data table refer to the minimum rotation angle, maximum rotation angle, an incremental angle, an active/inactive flag, and a null value, respectively.
- **proximity:** an index to the data table, indicating the specific proximity information for this constraint type.
- **assoc:** an index to the association table, indicating the entity's association to other specified entities.

- **contain:** an index to the containment table, indicating the entity's containment relationship with other entities.
- **bound_b:** an index to the bounding box table, indicating the top and bottom face of the entity's bounding box.
- **prox_b:** an index to the proximity box table, indicating the top and bottom face of the entity's proximity box.
- **bound_s:** an index to the bounding sphere table, indicating the entity's bounding sphere.

Data_Table, is an array structure which contains the actual values used to describe the design data and constraints. The `data_table` is a dynamic structure which size varies by the number of constraints specified as unique constraints by the user. This use of entity specification allows for the storage and retrieval of information pertinent to the constraint-based modeler. The `data_table` is composed of five elements, `value_a`, `value_b`, `value_c`, `flag_a`, and `flag_b`. The following is a discussion of each element:

- **value_a:** a float value which represents the minimum values for a specified field. This element is used for each of the character elements, as well as for the width proximity value in the relational structure.
- **value_b:** a float value which represents the maximum values for a specified field. This element is used for each of the character elements, as well as for the length proximity value in the relational structure.
- **value_c:** a float value which represents the incremental values for a specified field. This element is used for each of the character elements, as well as for the width proximity value in the relational structure. This field is not used for the area and volume characteristics.

- **flag_a:** an integer value which represents the active/inactive flag used to indicate the application of this constraint within the modeler.
- **flag_b:** an integer value which represents the replication and orthogonal continuity within a particular constraint. This field is not used for rotational attributes.

Association_Table, is a linked list array structure which contains the associations established between the entity and the other specified entities within the data structure. This list serves to represent how the element associates with each other element in the currently described world. The association_table is composed of three elements, entity, associate, and next. The following is a discussion of each element:

- **entity:** an index to the entity array structure, indicating the entity which the current entity is associated.
- **associate:** an integer value indicating the association with the entity specified in the entity field. A -1 (neg one) value represents an adverse or repel association, a 0 (zero) value represents no association and serves as the default value, and a 1 (one) value represents attraction association.
- **next:** an index to the next entity in the association table, which an association has been established.

Containment_Table, is a linked list array structure which contains the containment restrictions upon the entity. This list serves as the hierarchical tree structure governing the activity of the entity. The containment_table is composed of four elements, entity, type, cont_by, and next. The following is a discussion of each element:

- **entity:** an index to the entity array structure, indicating the entity.
- **type:** indicates the entity's spatial representation either as a solid or as a spatial.

- **cont_by:** an index to the entity array structure which indicates the entity which contains this entity.
- **next:** an index to the next entity in the Containment_table list.

Bounding Box Table, an array structure which contains the bounding box of the entity. This is important distinguishing characteristic of the solid entity as it establishes the link between the entity and the constraints. The bounding box is represented by the four points which define the base, and the four points which define the top. The representations, box_base and box_top, provide a static array for this information.

- **box_base:** an array of four elements, which represent three-dimensional Cartesian coordinate values of the four points of the base. A double value is stored for each value x, y, and z.
- **box_top:** an array of four elements, which represent three-dimensional Cartesian coordinate values of the four points of the top. A double value is stored for each value x, y, and z.
- **visible:** an integer value used to determine whether or not to display the bounding box representation.

Proximity Box Table, an array structure which contains the proximity box of the entity. This is important characteristic of an entity and is determined by the specification of a proximity distance which is stored in the data table. The pre-calculation of this offset provides an efficient method of maintaining the data during three-dimensional manipulation, and therefore limits the recalculation which would be required for every movement of the object. The proximity box is represented by the four points which define the base, and the four points which define the top. The representations, prox_base and prox_top, provide a static array for this information.

- **prox_base:** an array of four elements, which represent three-dimensional Cartesian coordinate values of the four points of the base. A double value is stored for each value x, y, and z.
- **prox_top:** an array of four elements, which represent three-dimensional Cartesian coordinate values of the four points of the top. A double value is stored for each value x, y, and z.
- **visible:** is not used.

Bounding Sphere Table, an array structure which contains the bounding sphere of the entity, is also provided. The bounding sphere is utilized to determine candidate entities for further intersection calculation. This method of proximity checking allows for the rapid elimination or determination of possible intersecting object.

- **cent_x:** a value indicating the x value of the centroid of the entity.
- **cent_y:** a value indicating the y value of the centroid of the entity.
- **cent_z:** a value indicating the z value of the centroid of the entity.
- **radius:** a value indicating the radius of the sphere.

The establishment of the tables, allows the algorithms to execute in a more efficient manner. Pre-calculation of data which is continuously utilized provides an effective solution to the exponential loss in performance due to the number of three-dimensional calculations required. In order to aid the programmer in visualizing and recording this information, the internal constraint data base can be written to an external file. This file provides the programmer with all of the user specified constraints pertinent to a specific entity, but also indicates the internally determined information as well. As a point of illustration, figure 112 provides a complete example of the constraint data structure for a cubic entity created in the fourth quadrant of the x-y plane.

Constraint Set for entity 1:

Entity: 1

fromobj 0
 name Obj1
 type 1

		Data Table:					
		idata	value_a	value_b	value_c	flag_a	flag_b
mobility	1						
width	9	9)	24.00	72.00	6.00	1	0
length	10	10)	48.00	96.00	6.00	1	0
height	11	11)	72.00	96.00	6.00	1	0
area	12	12)	125.00	250.00	0.00	1	1
volume	13	13)	0.00	0.00	0.00	0	0
xrot	14	14)	45.00	90.00	5.00	1	0
yrot	15	15)	45.00	90.00	5.00	1	0
zrot	16	16)	45.00	90.00	5.00	1	0
prox	17	17)	12.00	12.00	12.00	1	0

Association Table:

		iassoc	entity	associate	next
assoc	1	1)	1	0	3
		3)	2	1	6
		6)	3	-1	-1

Containment Table:

		icont	entity	type	contby	next
contain	1	1)	-1	1	-1	-1

Bounding Box Table:

		ibbox	box_base			box_top			
bound_b	1	1) ibase	x	y	z	itop	x	y	z
		0)	-168.00	-144.00	0.00	0)	-168.00	-144.00	96.00
		1)	-168.00	-90.00	0.00	1)	-168.00	-90.00	96.00
		2)	-102.00	-90.00	0.00	2)	-102.00	-90.00	96.00
		3)	-102.00	-144.00	0.00	3)	-102.00	-144.00	96.00

visible 1

Proximity Box Table:

		ipbox	prox_base			prox_top			
prox_b	1	1) ibase	x	y	z	itop	x	y	z
		0)	-180.00	-156.00	-12.00	0)	-180.00	-156.00	108.00
		1)	-180.00	-78.00	-12.00	1)	-180.00	-78.00	108.00
		2)	-90.00	-78.00	-12.00	2)	-90.00	-78.00	108.00
		3)	-90.00	-156.00	-12.00	3)	-90.00	-156.00	108.00

Bounding Sphere Table:

		ibsphere	cent_x	cent_y	cent_z	radius
bound_s	1	1)	-135.00	-117.00	48.00	180.53

Figure 112: An example of the internal constraint database for an entity.

4.2 Algorithmic Outline of Required Procedures

The implementation of the constraint-based solid modeler follows the form of the functional characteristics as described in Chapter III. To implement the internal workings and capabilities of the modeler, several additions and modification to the existing educational modeler, MacMod844, were performed. To illustrate the implementation of the constraint operations to the program, three main areas, knowledge specification, entity generation, and constraint modeling, are presented in order to clarify new procedures and modifications to existing procedures which were required. The following sections elaborate on the general requirements for implementation, where and how the requirements are composed within the existing program, and a general algorithmic outline of the process of constraint-based modeling with the procedures completed.

4.2.1 Knowledge Specification

As stated in Chapter III, section 3.4, the first of four main capabilities of the constraint-based solid modeler is to provide the ability to specify the design knowledge applicable to user definable three-dimensional entities. What this entails is the ability to input the knowledge specifications. This section deals with the user interface and the dialog handler required to input the design knowledge into the system specifications which are required for the generation of three-dimensional entities with constraints and includes the algorithmic outline for implementing these features.

4.2.1.1 User Interface

To facilitate the interaction between the user and the constraint modeler, several modification to the existing MacMod844, program were be made including additions to the menu selection routines, and the iconic interface. The changes to display the menu items and

icons were relatively minor and are not presented formally here. However, the interface between the dialog handler and the user is critical to the operation of the program and is presented. This interface is provided in the *x_command* function of MacMod844. To facilitate the proper activation and interface between the user and the dialog handler, the icon selection must be added accordingly. The outline below and the diagram (figure 113) indicate the process which was accommodated.

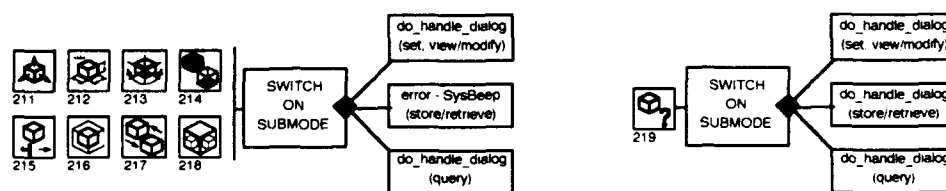


Figure 113: Illustration of the interface calls to properly invoke the dialog handler.

The following outline describes the additions to *x_command* which were made in order to facilitate the user interface for the knowledge specification/knowledge manipulation features of the constraint-based solid modeler:

- 1) Determine the selected command by performing a switch on the activIcon. This will determine the case of the selected mode the user desires to perform.
- 2) The active icon and the submode determines the parameters which will be passed to the constraint dialog handler.
- 3) Determine the submode selected (this is from the mode menu - constraints, and includes the options set, view/modify, store, retrieve, and query) and adjust parameters accordingly:

set - activates the modal dialog between the user and the system for the system specifications only.

view/retrieve - activates the modal dialog between the user and the system for all current specifications, entity and system.

store type - activates the modal dialog between user and system to store into memory the set of specifications selected. This is only allowed for the type definition icon.

retrieve type - activates the modal dialog between user and system to retrieve from memory the set of specifications selected. This is only allowed for the type definition icon.

query - activates the information (static) dialog indicating the particular information about the entity or system specifications. Note when selected via the type definition icon, a complete list of all parameters will be presented.

- 4) Make the call to the `do_dialog_handler` with the proper settings.

4.2.1.2 Dialog Handler

To provide the proper dialog routine calls, a dialog handler is included to aid the user. The dialog handler facilitates the link between the menu and icon selection routines and the dialog management routines. This relatively small function simply determines which dialog box the user requires, and whether or not the constraints apply to the system specifications or the entity specifications. The following is an outline of the procedure.

- 1) Determine the specification which is desired. This is performed by determining whether or not an entity has been selected. If an entity was selected, the specifications for that particular entity will be called upon, otherwise, the system specifications will be used.

- 2) Process the selection by determining which case is appropriate for this call. This is determined by the mode the user has selected, set, view/modify, store, retrieve, or query. Depending upon the case, one of the following options will be performed.
 - a. Set the system specifications accordingly. This is accomplished by invoking a modal dialog for the active icon and indicating that the systems constraints are to be used.
 - b. Set the entity specifications accordingly. This is accomplished by invoking a modal dialog for the active icon and indicating the specific entity's constraints are to be used.
 - c. Store the constraint specifications by invoking the storage function and providing the constraint name which is to be stored.
 - d. Retrieve the constraint specifications by invoking the retrieval function and providing the constraint name which is to be retrieved.
 - e. Perform a query on the selection by invoking a do_query function.
- 3) Return to the calling function.

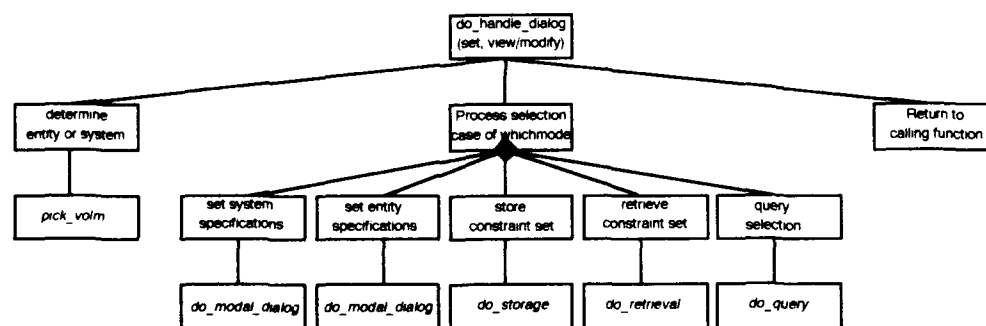


Figure 114: Illustration of the dialog handler.

4.2.2 Entity Generation

The second of the three main areas of the constraint-based solid modeler is to provide the ability to generate, view, modify, store, and retrieve the entity along with its constraints. This section outlines the algorithm required to create the three dimensional representation utilizing the solid data structure presented in section 4.1.1 as well as the constraint data structure presented in section 4.1.2. The following algorithm is based on the current operations of MacMod844.

4.2.2.1 Generation within Dimensional Constraints

The following algorithm was incorporated as part of the MacModeler routines, and is invoked when the constraint mode of operation is active while in the extrusion mode.

- 1) Calculate the three-dimensional coordinate value of the point selected.
- 2) Determine if first point selected lies within any existing solid entities. (Point_inside_solid) If so, reject attempt to create a solid entity.
- 3) Set adjustment factor to full value (1.0) for rectilinear entities, and half value (0.5) for all other primitive entities. (This allows for the difference in creation techniques of primitives.)
- 4) Check the set height value for compliance with the allowable range of values:
 - a. If height set is greater than the allowable maximum, set height to maximum value.
 - b. If height set is less than the allowable minimum, set height to minimum value.
- 5) Set the generation threshold values for length and width:
 - a. Set the allowable minimum length to the adjusted Length Minimum.
 - b. Set the allowable maximum length to the adjusted Length Maximum.
 - c. Set the allowable minimum width to the adjusted Width Minimum.

- d. Set the allowable maximum length to the adjusted Width Maximum.
- 6) Calculate the screen point of the three-dimensional coordinate value and set and mark the results as the first point of the base.
- 7) Initialize the horizontal and vertical indexes to zero. Initialize the beginning point array, the ending point array, the polygon point list, and the created array, allocating sufficient memory space.
- 8) Assign the beginning point of the initial object the value of the first point. The beginning point array carries the first point for each object based upon the horizontal and vertical indexes.
- 9) Set the old point equal to the first point; this sets the last mouse position.
- 10) Set the penmode to Xor in preparation for rubberbanding the primitive.
- 11) Initialize the created array indicating no objects have been drawn.
- 12) While the mouse button is depressed, perform the following steps to rubberband the primitive object interactively including replication if required:
 - a. Read the new mouse location and determine if its position has changed.
 - b. If this is a new location, perform the following:
 1. If the replication attribute of the dimensional constraint has been set, adjust the horizontal and vertical indexes according to the following:
 - (a) Calculate the three-dimensional coordinate value of the beginning point and the new point adjusting for the reference plane.
 - (b) If the horizontal distance between the new location and the current beginning point is equal to the maximum allowable length, determine if the horizontal movement is away from the beginning point.
 - (c) If the horizontal movement is away from the previous point, that is the direction of movement is moving away from the beginning point, perform the following:

- (1) increment the horizontal index
 - (2) set the beginning point for each replicated object (each new instance created by incrementing the horizontal index) offsetting the horizontal value by the length proximity value and if non-rectilinear primitive, adding the allowable length.
- (d) If the horizontal movement is toward the previous point, that is the direction of movement is moving to the beginning point, perform the following:
- (1) for each replicated object (each instance of the current horizontal index) erase the mark designating the beginning point, erase the polygon, and set the created index to false.
 - (2) decrement the horizontal index.
- (e) If the vertical distance between the new location and the current beginning point is equal to the maximum allowable width, determine if the vertical movement is away from the beginning point.
- (f) If the vertical movement is away from the previous point, that is the direction of movement is moving away from the beginning point, perform the following:
- (1) increment the vertical index
 - (2) set the beginning point for each replicated object (each new instance created by incrementing the vertical index) offsetting the vertical value by the width proximity value and if non-rectilinear primitive, adding the allowable width.
- (g) If the vertical movement is toward the previous point, that is the direction of movement is moving to the beginning point, perform the following:

- (1) for each replicated object (each instance of the current vertical index) erase the mark designating the beginning point, erase the polygon, and set the created index to false.
 - (2) decrement the vertical index.
2. For each horizontal object (0 to horizontal index) and for each vertical object (0 to vertical index) perform the following:
 - (a) Calculate the three-dimensional coordinate value at the beginning point and the new point adjusting for the reference plane.
 - (b) Determine whether the new width is less than the minimum allowable width, within the minimum and maximum allowable width, or greater than the maximum allowable width.
 - (c) Determine whether the new length is less than the minimal allowable length, within the minimum and maximum allowable length, or greater than the maximum allowable length.
 - (d) Set the working zone according to the following:
 - zone 1: Width is less than the minimum allowable width AND Length is less than the minimum allowable length.
 - zone 2: Length is within the allowable tolerances for length AND (Width is less than the minimum allowable width OR greater than the maximum allowable width.)
 - zone 3: Width is within the allowable tolerances for width AND (Length is less than the minimum allowable length OR greater than the maximum allowable length.)
 - zone 4: Length is within the allowable tolerances for length AND Width is within the allowable tolerances for width.

zone 5: (Length is greater than the maximum allowable length AND Width is greater than the maximum allowable width) OR (Length is greater than the maximum allowable length AND Width is less than the minimum allowable width) OR (Length is less than the minimum allowable length AND Width is greater than the maximum allowable width.)

3. If the object has been created (created index is true) then rubberband the object according to the following zone restrictions:
 - (a) If the location zone is 1 or 5, No rubberbanding is allowed.
 - (b) If the location zone is 2, Draw the old polygon, adjust the vertical value of the objects endpoint accordingly, and draw the new polygon.
 - (c) If the location zone is 3, Draw the old polygon, adjust the horizontal value of the objects endpoint accordingly, and draw the new polygon.
 - (d) If the location zone is 4, Draw the old polygon, adjust both the horizontal value and the vertical value of the objects endpoint accordingly, and draw the new polygon.
4. If the object has not been created (created index is false) then rubberband the object according to the following zone restrictions:
 - (a) If the location zone is 1, 2, 3, or 5, No rubberbanding of object is allowed.
 - (b) If the location zone is 4, set the creation index to true, set the endpoint of the entity equal to the new point, draw a mark for the beginning point, and draw the new polygon.
5. Set the old point equal to the new point thus preserving the current location as the previous location.

- 13) Upon the release of the mouse button for each horizontal object (0 to horizontal index) and for each vertical object (0 to vertical index) perform the following:
 - a. determine if the polygon primitive has been created. If created index is false then no entity will be generated,
 - b. If the horizontal index, the vertical index, and the created index are all set to zero, return to the event manager, no object is created.
 - c. If created is true then draw the new polygon and delete mark for first point.
This prepares the graphics screen for the generation of the solid elements.
- 14) Set the pen mode to copy.
- 15) For each horizontal object (0 to horizontal index) and for each vertical object (0 to vertical index) perform the following:
 - a. check the area of the polygon primitive. If the area of the polygon is negative, reverse the direction of the segments which compose the polygon.
 - b. create the three dimensional representation of the solid entity by extrusion of the base polygon primitive stored in the temporary polygon array. Note these polygon primitives satisfy the dimensional constraints set in the system specifications.
 - c. store the constraint values into the Constraint data structure for the entity by incrementing the constraints data structure index, assigning the entity index to the object, and setting the values for each of the constraint fields to that of the system constraints. This sets the entities individual constraints for later use.
 - d. store the bounding box data for the entity in the corresponding Base_Array indicating the beginning point, the ending point, the height, and the primitive type.
- 16) Free the following temporary arrays, beginning point array, ending point array, created array, and the polygon points list array.

- 17) Plot the three-dimensional view of the solid data structure, thus graphically indicating the results of the generation.
- 18) Return to the system manager.

4.2.3 Constraint Modeling

The last of the capabilities of the constraint-based solid modeler presented, is to provide the ability to manipulate the three-dimensional entities in a manner which is consistent with the behavioral characteristics dictated by the entity specifications. This, the largest requirement of the implementation, requires the adaptation of existing procedure to provide the required constraints upon the generation and manipulation of the entities. This section will handle the methods of modeling and constraining the modeling process to the constraint/attributes of the entity. Each of the main constraint/attributes is presented and includes the algorithmic outlines for implementation.

4.2.3.1 Entity Manipulation With Constraints

The following general algorithm illustrates the basic manipulation of an entity within the constraints set in the modeler. References are made to routines which follow later in this chapter. This algorithm handles the basic interaction with the user and will handle the interaction between primary and secondary entities.

- 1) Construct the two-dimensional representation of the selected entity adding it to the rubberbanding list.
- 2) Begin the rubberbanding sequence by setting the penmode to xor, and reading the new location of the mouse.
- 3) While the mouse button is still depressed, perform the following steps:
 - a. Get the current location of the mouse from the system.

- b. If the current location differs from the previous location, the following is to be performed:
 - 1. Check constraint satisfaction for the anticipated manipulation. The algorithm for constraint satisfaction is presented below.
 - 2. If movement is allowed, perform the following steps to represent the image movement.
 - (a) Loop through the rubberbanding list and draw all old two-dimensional representations.
 - (b) Loop through the rubberbanding list and update the representations.
 - (c) Loop through the rubberbanding list and draw the new two-dimensional representations.
 - 3. Reset the rubberbanding index to the original selected entity.
- c. Set the previous point equal to the current point.
- 4) Reset the pen mode to patCopy in preparation for redrawing the final three-dimensional representation.
- 5) Redraw the three-dimensional representation of the solid elements.

4.2.3.2 Constraint Satisfaction

The following algorithm provides the main interactive constraint satisfaction between the interaction of entities. This accounts for proximity constraints, associative constraints, and the manipulation of the entities within those constraints. This routine is called when geometric editing features, such as translation, rotation and scale, are executed by the modeler and is capable of coordinating the resulting interaction caused by such editing of the primary entity.

- 1) If the entity is the primary entity (entity being manipulated) or the entity has a mobility constraint set to free and the entity is not a space, perform the following,

otherwise, the manipulation of the entity is not allowed, control is returned to the calling routine.

- a. Check the rubberbanding list to verify the entity is not currently present. If the entity is not listed, add the entity to the list and perform the following to prepare the entity to be rubberbanded:
 1. Construct the two-dimensional representation of the entity.
- 2) Transform the solid representation of the entity accordingly, including the bounding box representation.
- 3) Check for proximity detection and proximity interference, storing those entities in the appropriate table, proximity detection table, and/or proximity interference table. (This algorithm is listed below.)
- 4) If the proximity detection set and the proximity interference set are both empty, movement is allowed.
- 5) Check the proximity detection list to identify those entities which proximity has been noted. If the entity in the list is not contained in the proximity interference list perform the following association adjustments according to the constraints of the entities.
 - a. If the entity in the proximity detection set has no association with the primary entity, update the two-dimensional representation of the entity and return control to the calling routine.
 - b. If the secondary entity has a mobility characteristic set to fixed, one of the following operations must be performed.
 1. If the association between the primary and secondary entity is attract, snap the primary entity to the secondary entity as follows.
 - (a) While there is no interference detected, incrementally transform the entity and update the bounding box.

- (b) When interference is detected, restore the primary entity to its previous position, update the bounding box, and update the two-dimensional representation of the entity.
- 2. If the association between the primary and secondary entity is repel, the primary entity is not allowed to execute the desired transformation, the following corrective measures must be performed:
 - (a) Instruct the calling routine that no movement is allowed.
 - (b) Restore the position of the entity and the bounding box representation, to the condition prior to the transformation conducted in step 2.
 - (c) Return to the calling routine.
- c. If the secondary entity has a mobility characteristic set to free, one of the following operations must be performed:
 - 1. If the association between the primary and secondary entity is attract, snap the secondary entity to the primary entity as follows.
 - (a) While there is no interference detected, incrementally transform, in the opposite direction, the secondary entity and update the bounding box.
 - (b) When interference is detected, restore the secondary entity to its previous position, update the bounding box, and update the two-dimensional representation of the entity.
 - 2. If the association between the primary and secondary entity is repel, the secondary entity must check for valid transformation by calling (recursively) the constraint checking routine and determine constraint satisfaction for secondary entity movement.
 - (a) If, after return from the recursive call, movement is not allowed, the following corrective measures must be performed on the entity:
 - (1) Instruct the calling routine that no movement is allowed.

- (2) Restore the position of the entity and the bounding box representation, to the condition prior to the transformation conducted in step 2.
 - (3) Return to the calling routine.
- (b) If, after return from the recursive call, movement is allowed, continue with the procedure.
- 6) If the interference set contains entities which the manipulation of the current entity effects, loop through each entity in the set, recursively calling the constraint satisfaction algorithm, and check the constraint satisfaction for subsequent secondary manipulation of the entity.
- 7) If, after return from the recursive call, movement is not allowed, the following corrective measures must be performed on the entity:
 - a. Instruct the calling routine that no movement is allowed.
 - b. Restore the position of the entity and the bounding box representation, to the condition prior to the transformation conducted in step 2.
 - c. Return to the calling routine.
- 8) If, after return from the recursive call, movement is allowed, instruct the calling routine that movement is allowed, update the two-dimensional representation of the entity, and return control to the calling routine.

4.2.3.3 Detection and Interference Determination

The basic structure of this algorithm follows a hierarchical sequence of determining 1) proximity detection, the intersection or penetration of one proximity zone and another proximity zone, and 2) proximity interference, the intersection or penetration of a proximity zone and an entity. The distinction between the two becomes important when satisfying the constraints for proximity, association, and interaction between spatial entities (solids and

spaces), and is used to perform the appropriate operations to meet the constraints of the modeler.

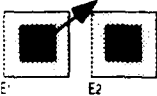
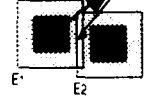
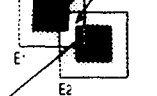
	 <p>No Proximity Detection</p> <p>No Proximity Interference</p>	 <p>Proximity Detection</p> <p>No Proximity Interference</p>	 <p>Proximity Detection</p> <p>Proximity Interference</p>
Association (E1 and E2)			
No Association E2 Fixed	E1 free to transform	E1 free to transform	E1 Retreats to original position
No Association E2 Free	E1 free to transform	E1 free to transform	E2 transforms(repels) until detection is clear.
Attract with E2 Fixed	E1 free to transform	E1 Snaps to E2	E1 Retreats to original position
Attract with E2 Free	E1 free to transform	E2 Snaps to E1	E2 transforms(repels) until detection is clear.
Repel with E2 Fixed	E1 free to transform	E1 Retreats to original position	N/A
Repel with E2 Free	E1 free to transform	E2 transforms(repels) until detection is clear.	N/A

Figure 115: Proximity/Association Resultant Matrix.

In order to perform this detection and notification in an efficient manor, this algorithm employs a hierarchical search and testing method. This algorithm utilizes preliminary, and computationally efficient, methods of determining the entities which can be trivially rejected from further consideration. This allows the more computationally expensive test to be performed only on the select few entities which have the highest probability of interference with the operations of another entity. There are three levels of testing which will be performed successively on each successful candidates; bounding sphere proximity, point within an entity, and edge intersection. The general algorithm is presented below and includes a brief algorithmic description of each test.

- 1) Given the primary entity, check all remaining secondary entities for intersection between the bounding spheres. This is accomplished accordingly:
 - a. Determine the distance between the centroid of the bounding sphere for the primary entity with that of the secondary entity.
 - b. Determine the separation distance by subtracting the radius of the each entity, primary and secondary, from the overall distance determined in 1.a.
 1. If the separation distance is negative, add that entity to the candidate list, and continue.
 2. If the separation distance is positive, the two entities do not intersect. discard the secondary entity from further consideration.
- 2) For each secondary entity in the candidate list, check for further intersection between proximity spaces of the primary entity and the candidate entity by performing the two intersection tests.
 - a. First determine whether or not any vertex of the secondary entity's proximity space is inside the proximity space of the primary entity, or whether or not any vertex of the primary entity's proximity space is inside the proximity space of the secondary entity. This is the point in an object test and must be applied to both cases.
 1. For each vertex of the proximity space of the secondary entity, determine whether or not the vertex lies on the inside of each face of the proximity space of the primary entity.
 - (a) If any vertex lies on the inside of all of the faces of the proximity space of the primary entity, proximity detection is noted and the secondary entity is added to the proximity detection list. Further testing of the secondary entity is not required.

- (b) If all vertices of the proximity space for the secondary entity fail this test, the second portion of the test must be performed, that is, the two entities must be tested again with the roles reversed.
- 2. For each vertex of the proximity space of the primary entity, determine whether or not the vertex lies on the inside of each face of the proximity space of the secondary entity.
 - (a) If any vertex lies on the inside of all of the faces of the proximity space of the secondary entity, proximity detection is noted and the secondary entity is added to the proximity detection list. Further testing of the secondary entity is not required.
 - (b) If all vertices of the proximity space for the primary entity fail this test, additional tests must be performed to determine other types of intersection.
- b. If the secondary entity fails the initial test, additional testing must be performed. The secondary entity must be tested for edge intersection. Edge intersection will determine whether or not the edge of the proximity space for the secondary entity intersects with the proximity space for the primary entity.
 - 1. For each edge of the proximity space of the secondary entity, determine whether or not the edge intersects any two faces of the proximity space of the primary entity.
 - (a) If the midpoint of the two intersecting points lies on the inside of the proximity space of the primary entity, proximity detection is noted and the secondary entity is added to the proximity detection list. Further testing of the secondary entity is not required.

- (b) Otherwise, the secondary entity fails this test and additional tests must be performed to determine the whether or not the last type of intersection is valid.
- 3) The conclusion of step 2 will result in a list of all secondary entities in which proximity detection has been determined.
- 4) For each secondary entity in the proximity detection list, check for intersection between proximity spaces and the entities of each the primary entity and the secondary entity by performing the three intersection tests indicated in step 2 again. This secondary testing must test for interference between the entities themselves and the proximity space limitation of the entity. Each test must be accomplished twofold, 1) for intersection between the secondary entity's proximity space and the primary entity itself, and 2) for the primary entity's proximity space and the secondary entity itself. (Keeping this in mind step two will not be repeated here.)
- 5) The conclusion of step 4 will result in a list of all secondary entities in which proximity interference has been determined. and concludes the intersecting constraint algorithm.

CHAPTER V

C•MOD APPLICATIONS

The interactive simulation of solid and spatial entities, as previously mentioned, is critical to the success of a constraint-based solid modeler. To illustrate the orchestration and interaction between both spatial and solid entities, several brief examples of the prototypical application C•Mod are presented. The purpose of this chapter is to provide a demonstration of the working capabilities of C•Mod and to demonstrate its usefulness as a conceptual design tool. Three main areas are explored to provide this demonstration, conceptual design of an architectural element, conceptual design of building components, and finally, conceptual design within a design space.

5.1 Conceptual Design of an Architectural Element

Representing solid entities within a solid modeler provides the basic, and most fundamental method of visualizing architectural elements. C•Mod provides a means of extending this basic capability by allowing design knowledge to control, or constrain, the generation and manipulation of the elements in a three-dimensional environment. To illustrate the fundamental capabilities of the constraint-based solid modeler with regards to this interactive process, the creation of an architectural element is sequentially presented. The generation and manipulation of the pedestal demonstrates the interactive nature of modeler, and illustrates that the system is capable of supporting the lower level decisions of constraint satisfaction.

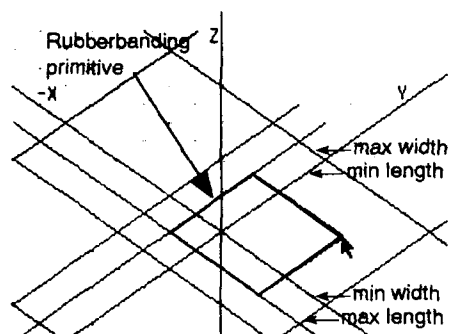


Figure 116: Generation of an entity within dimensional constraints.

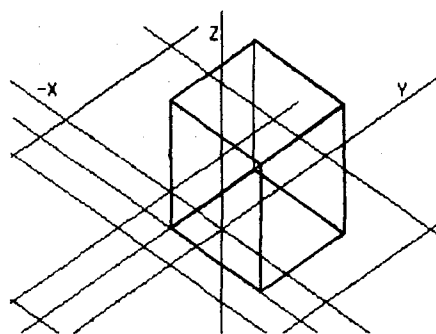


Figure 117: Results illustrating the generated entity.

The modeling process begins with the specification of the constraints applicable to this architectural element. Although not shown, a minimum and maximum height, length and width have been specified for the entity. Figure 116 illustrates the generation of an entity within those dimensional constraints. The construction lines have been provided to illustrate the dimensional restrictions. The results of this interactive generation process are illustrated in a wire-frame image in figure 117.

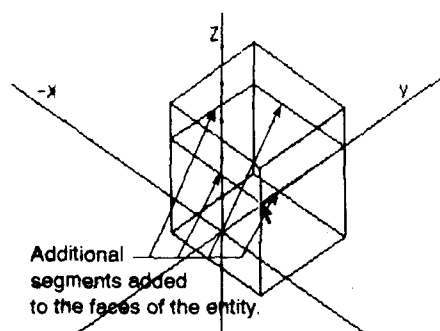


Figure 118: Topological editing of the entity.

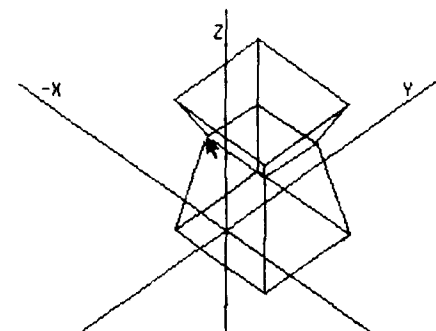


Figure 119: Geometric editing of the inserted segments.

The creation of a cube provides the designer with the basic foundation with which to model the desired pedestal. Utilizing the topological editing operations of the modeler, the

designer can interactively sculpt the cube into the shape which satisfies a particular requirement (figure 118). Further geometric editing of the newly constructed faces, allows the pedestal to take form. Figure 119 illustrates this operations on the entity.

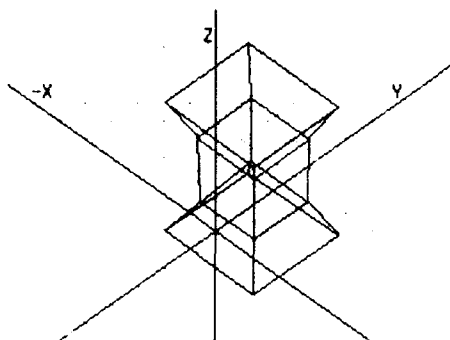


Figure 120: Subsequent topological and geometric editing to create an architectural element.

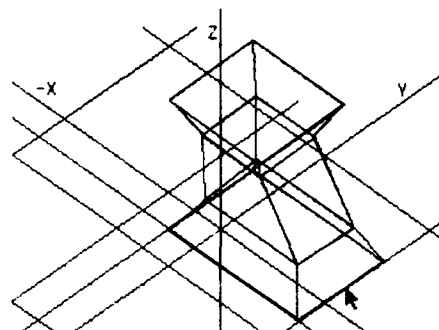


Figure 121: Further geometric editing of a face of the architectural entity. Note translation of a face limited to the dimensional constraints.

By successively editing the model in this manner, the user can quickly create an architectural element which is within the desired limitations of the dimensional requirements without being overly concerned with the data itself (figure 120). With the element created to meet the basic requirements, further manipulation of the entity illustrates the advantage of interactively modeling constraints. Further molding of the pedestal is allowed to be performed within the dimensional constraints specified (figure 121). This allows the designer the flexibility of modeling a new form which satisfies the maximum limitations of the dimensional constraints. The dimensional constraint construction lines were provided to illustrate the maximum limitation, or bounds, of allowable interactive manipulation. This figure illustrates that any topological level, in this case a face, can be edited interactively, and such interactive editing is constrained to the design knowledge which is internally represented. Subsequent geometric editing of the capital results in an architectural element which satisfies the intent of

the design. Figure 122 illustrates the completed pedestal. As described in Chapter I, the results produced in this manner satisfy the design objectives and requirements.

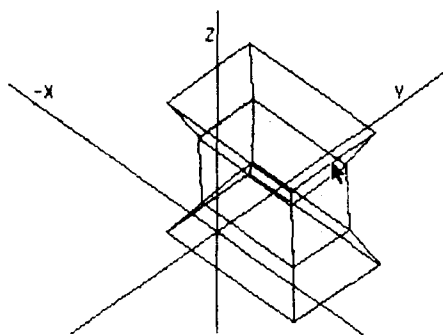


Figure 122: Subsequent geometric editing within dimensional constraints.

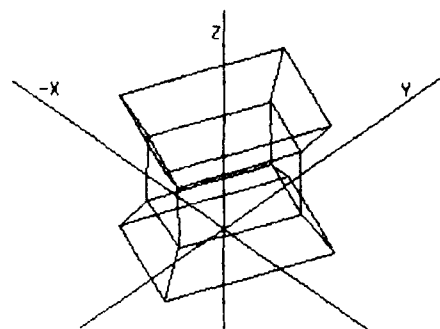


Figure 123: Rotation of the element on the x-y plane.

As previously mentioned, the representation of the dimensional data is not restricted to the orthogonal plane. Rotation of the pedestal (figure 123) will not disrupt or distort either the actual dimensional data, or the constraints which are imposed on that dimensional data. Figure 124 illustrates that subsequent scaling of the entity will still be restricted to the dimensional limitations. This figure also provides a visualization of the bounding box which is used to store the dimensional data, and to ensure satisfaction with the dimensional constraints.

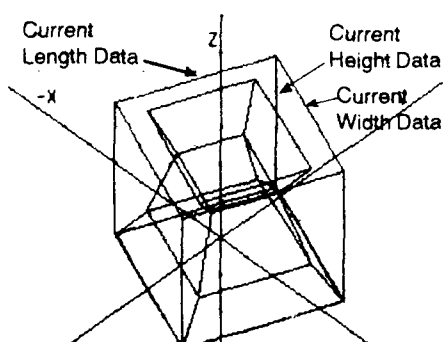


Figure 124: Illustration of the data represented after rotation.

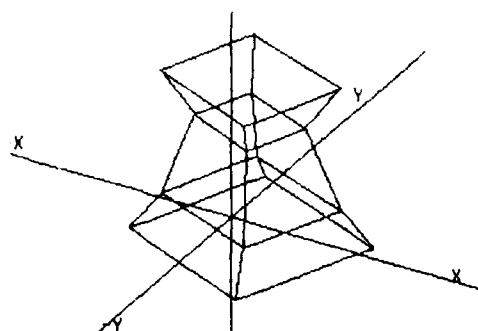


Figure 125: Illustration of completed architectural element modeled in C*Mod.

The results of the interactive solid modeling process within the limitations of design knowledge are provided in figures 125 and 126. The designer has created a pedestal which will perform as a solid entity, and which was created interactively within dimensional guidelines specified. By providing additional constraints upon the entity, such as mobility and relationship characteristics, the pedestal will behave in a manner which is consistent with its definition.

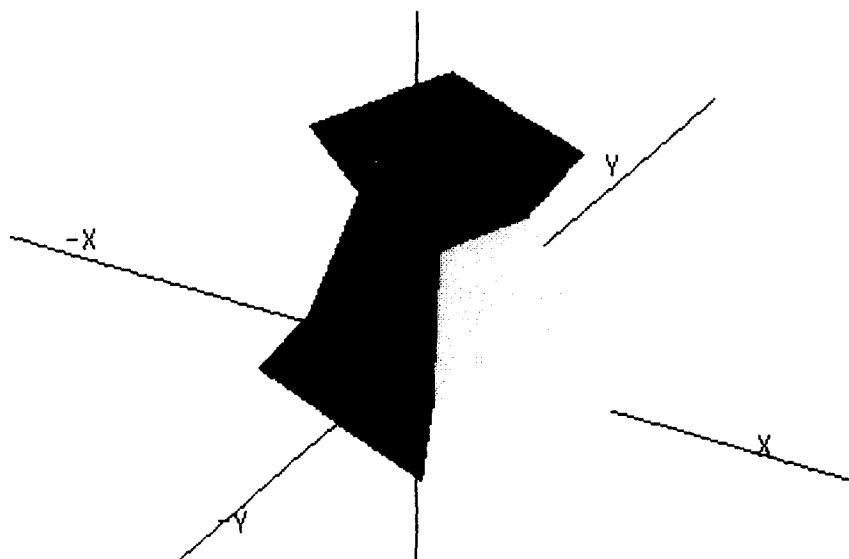


Figure 126: A rendered illustration of an architectural element, which was created utilizing the constraint operations of C•Mod.

5.2 Conceptual Design of Building Components

The previous section discussed the construction of an architectural element within the guidelines of design knowledge. Taking the process one step further, into interactive modeling of a conceptual design utilizing building components, the benefit of providing behavioral characteristics which constrain the interaction between these components can be clearly shown. This section explores the interaction between entities during the interactive modeling process

and provides an example of the use of a constraint-based solid modeler during the conceptual design process. The discussion centers around the development of a conceptual design in which three rooms are to be modeled as a building mass. Emphasis is placed on the interaction and behavior of the conceptual elements during the interactive process.

Three rooms have been modeled utilizing the void modeling operations of C•Mod. These rooms are modeled as solid entity with no floor or ceiling, and include a window. Figure 127 illustrates a wire-frame representation of these components. The scene can be interactively manipulated by selection of an object and translating it in three-space. Since none of the objects have any relationship or mobility restrictions placed upon them, interaction between solid entities results in subsequent translation, and therefore, allow solid objects to move other solid objects. Figure 128 illustrates this cascading, or ripple effect of translating room A along the x-y reference plane. Since C•Mod interactively models this effect, the user can visualize the consequences, or benefits, of editing the scene without being concerned over overlapping objects, or conflicting entities.

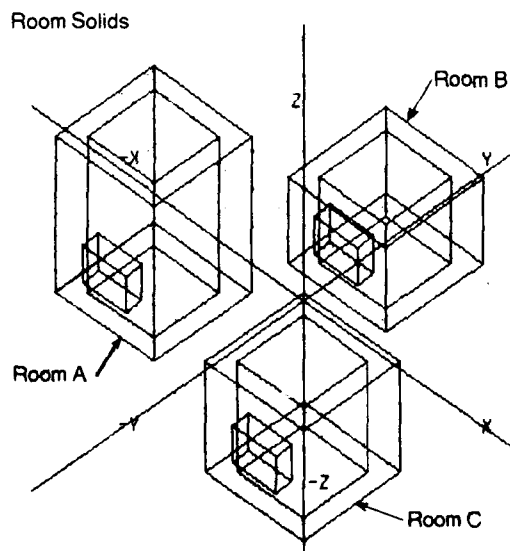


Figure 127: An illustration of three rooms created with the void modeling operations.

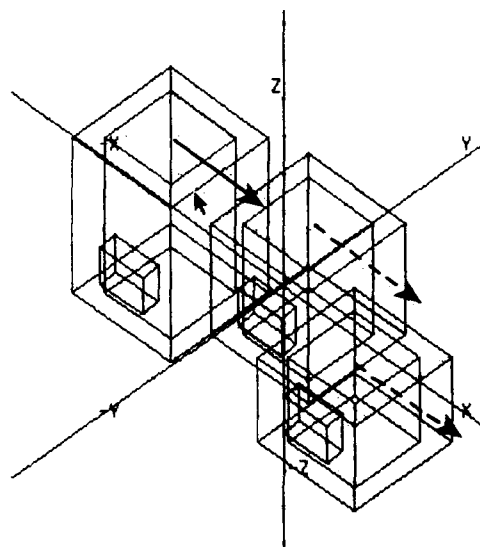


Figure 128: Geometric editing of room A, and subsequent translation of B and C.

Placement of the rooms, can be made quickly and efficiently, without the need for exact precision. This can be accomplished by determining the desired location of one of the rooms, and fixing its mobility in three-space. This allows other entities to be placed directly against it without the entity being displaced. Figure 129 illustrates the results of such interactive manipulation where room C was placed and fixed in three-space, and the other rooms were subsequently located.

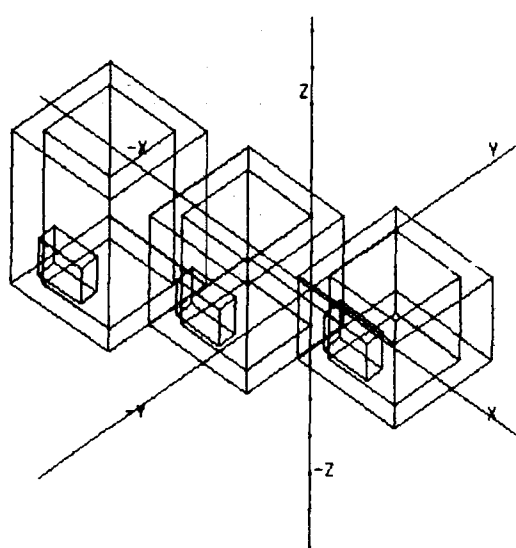


Figure 129: Final placement of three rooms constrained by the solidity of the entities.

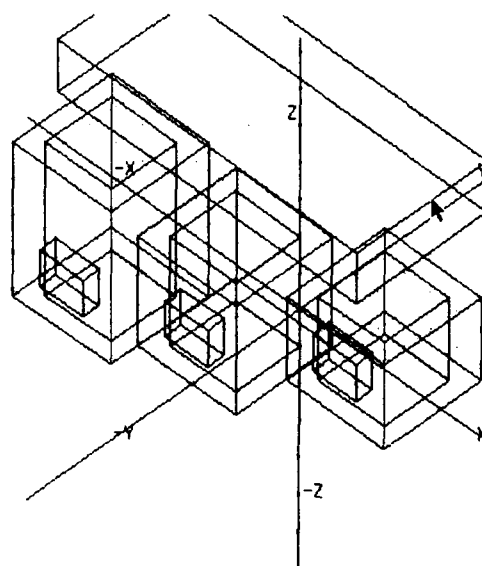


Figure 130: Generation and placement of a horizontal solid entity.

The conceptual design progresses further by generating a roof element with dimensional constraints attached (figure 130). Manipulation of the reference plane to place the roofing element over the rooms will allow element to be placed in a desirable position. Considering that room C has a fixed mobility, the roof element can be translated along the z axis to align the top of each of the rooms with the bottom of the roof. Since room C restricts any further movement of the roof, the user does not have to be concerned with precision. In

addition, rooms A and B do not require further alterations since their mobility is free, the translation of the roof results in the translation of the rooms (figure 131). Once again, the benefit of modeling solid characteristics are easily visualized.

Creating and duplicating similar entities can be tedious at times. Therefore, C*Mod provides the user with the ability to replicate entities when the generation process exceeds a dimensional boundary. An application of the replication operation is the generation of columns to support the roofing element. By selecting replication in the length direction, a series of columns can be created from a single generation sequence. Figure 132 illustrates the generation of three columns within dimensional constraints. Note that the construction lines for the replicated entities are not provided. The results of this multiple generation process are illustrated in figure 133. It is important to note that all three columns have identical set of constraint specifications. Thus not only the objects but the constraints are replicated as well.

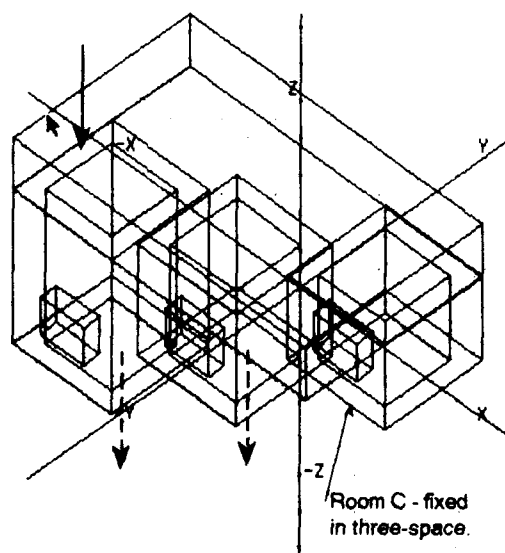


Figure 131: Geometric editing of the roof entity and resulting interaction with the rooms. Note room C is fixed and restricts any further movement of the roof.

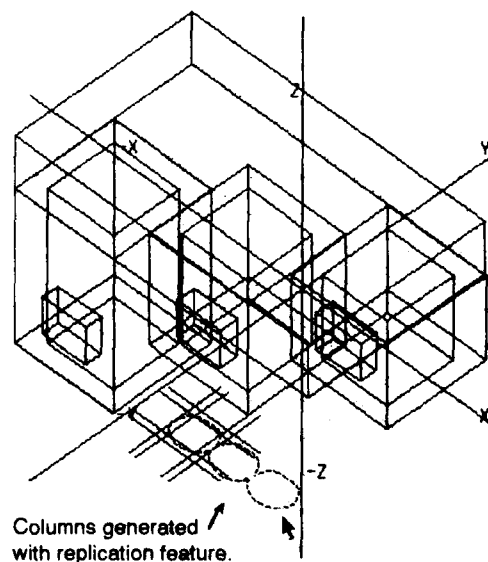


Figure 132: An illustration of element generation within dimensional constraints with the replication operations activated.

The conceptual design can be further enhanced by the proper placement of the columns (figure 134), and the modification of the roof. The roof can be manipulated through the use of the topological and geometrical editing features of the modeler. Additionally, since the roofing element has a height constraint, vertical translation will be limited to this upper limit (figure 134). This use of the dimensional constraints illustrates an efficient method of interactively modeling an entity.

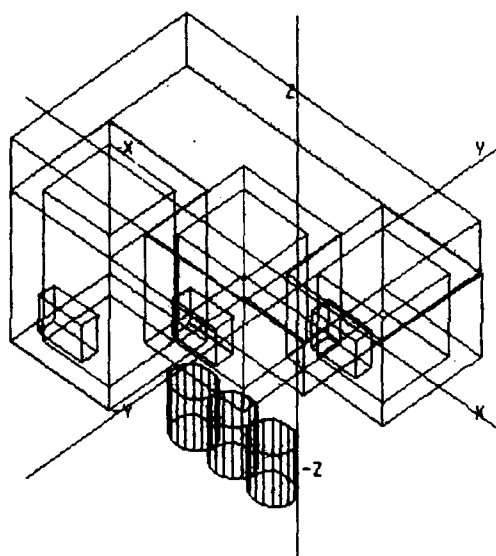


Figure 133: The results of multiple column generation by replication.

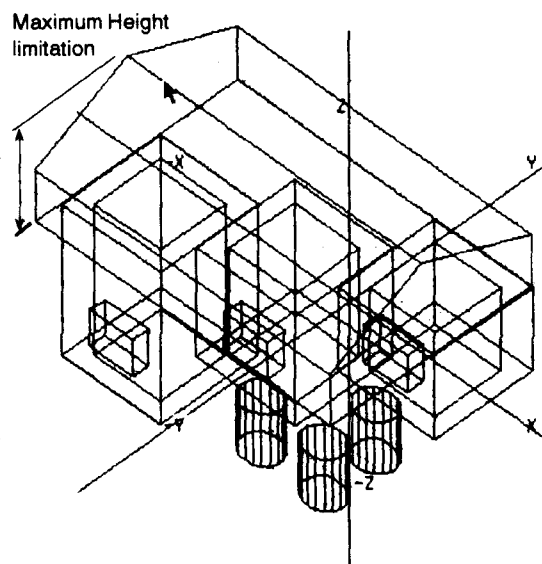


Figure 134: Topological and Geometric editing within dimensional constraints.

Utilizing the topological and geometric editing operations of the modeler, a complete hip roof can be generated to enhance the design (figure 135). Additionally, setting the mobility of the roof element to fixed and establishing an attract association with all objects of the column type will enable the columns to be snapped to the bottom of the roof. This establishes an attract association and will keep the columns and the roof together in the event that either are moved. Figure 136 illustrates the results of this attraction between the roof and the columns.

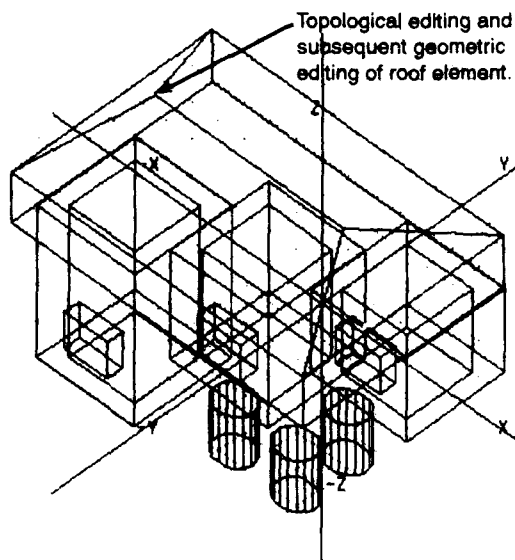


Figure 135: Continued topological and geometric editing of the roof element.

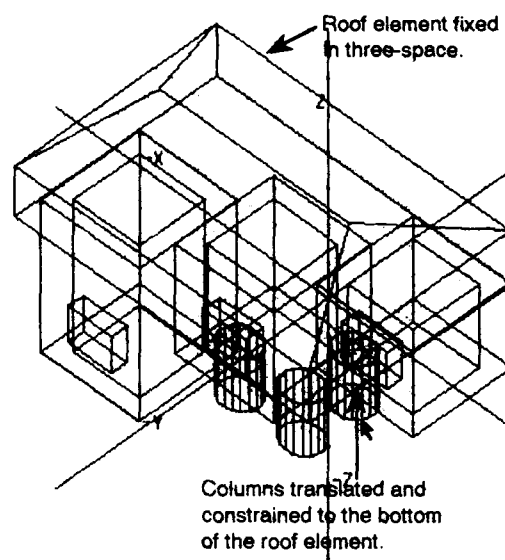


Figure 136: Translation of the columns constrained by the mobility of the roof.

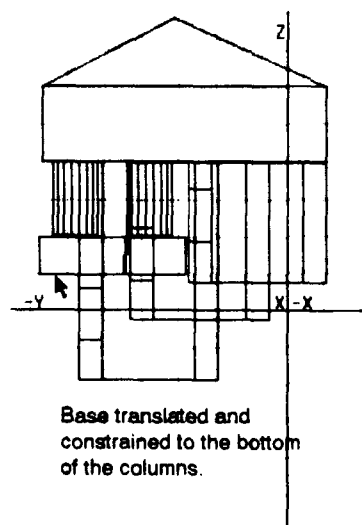


Figure 137: Elevation view of the placement of a base for the columns restricted by the mobility of the roof.

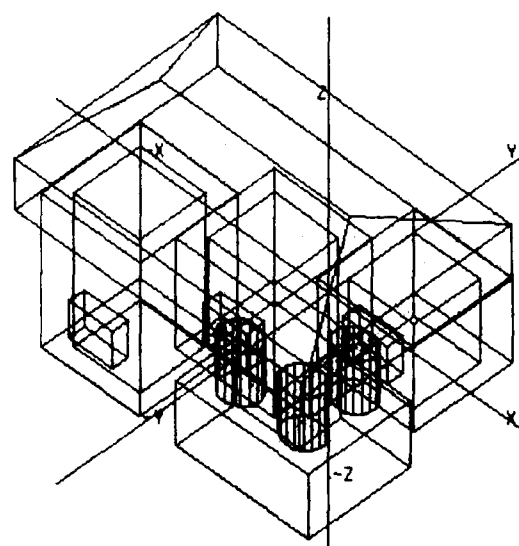


Figure 138: Results of the constraint-based interactive modeling utilized for a conceptual design.

To complete the conceptual design, the columns need to be attracted to a base to provide additional support. To illustrate that the modeler can perform the same operations, regardless of the reference plane, or view, figure 137 provides a side elevation in which a base is being modeled. The interactive nature of the modeler does not preclude the use of any view, thus, allowing interactive modeling in three-space from any available view.

Providing a means of realistically modeling design knowledge allows the conceptual design phase to respond to behavioral and physical requirements set by the user. Results of the interactive modeling process utilizing the constraints are illustrated in figures 138 and 139.

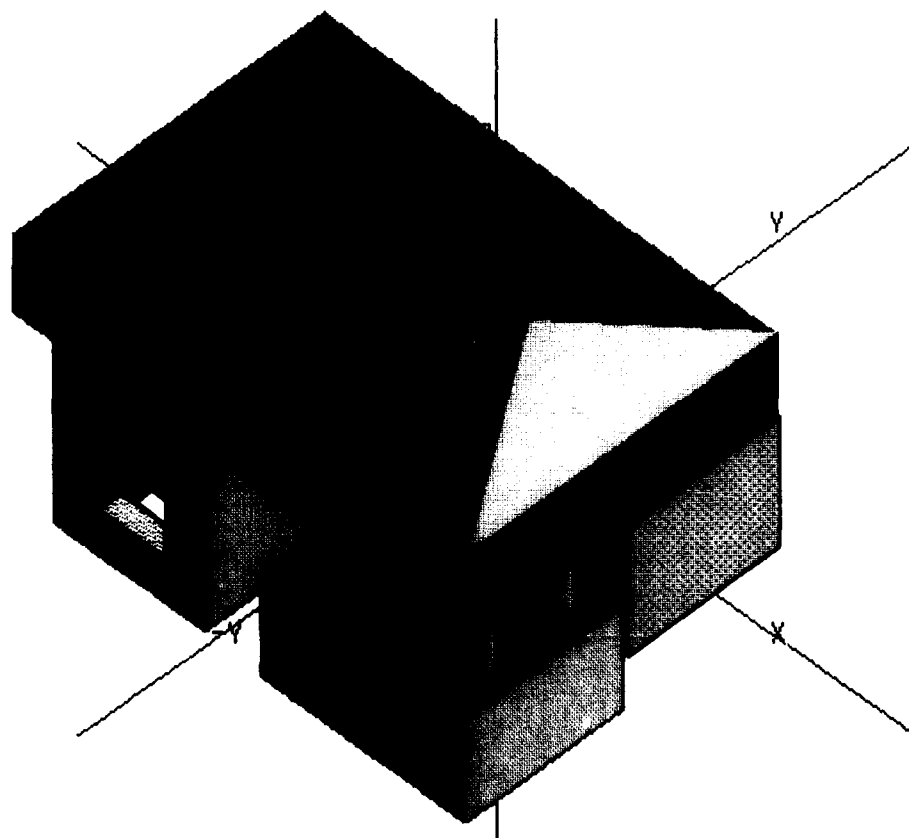


Figure 139: Rendered results of a schematic design solution utilizing C•Mod.

5.3 Conceptual Design Utilizing the Design Space

With the understanding of the constraint-based modeling applications at the entity generation level of conceptual design, and the entity interaction level of conceptual design, the final discussion of the application of the modeler provides a visualization of the concepts of interactive modeling within a realistic design space. This section illustrates the process of establishing a hypothetical design space in which solid entities representing spatial forms can be generated and manipulated within the constraints of the design space as well as the constraints imposed on the entities themselves. The results of this interactive constraint-based modeling illustrate the use of C•Mod as a conceptual design tool.

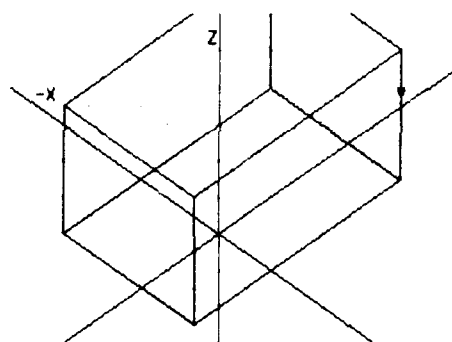


Figure 140: Creation of a design space within dimensional constraints.

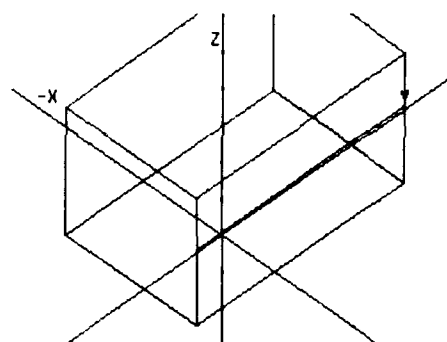


Figure 141: Modeling the design space interactively by inserting a segment in the face of the space.

To begin the process of interactive constraint-based solid modeling, a design space is created (figure 140). This allows a site to be modeled, with appropriate setbacks, height restrictions, and shadow/sunlight restrictions. Figure 141 above illustrates the result of the generation process, and the beginning of the interactive editing process. The design space has been generated with a proximity offset established in the width direction, and is used to illustrate a side yard setback within the allowable design space.

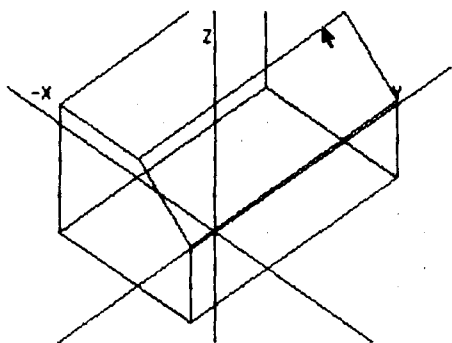


Figure 142: Establishing the shadow/sunlight restriction upon the design space.

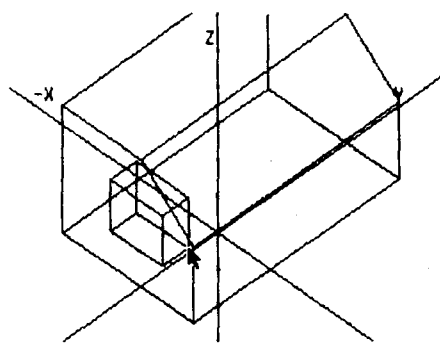


Figure 143: The generation of a solid entity within the design space.

Further geometric editing of the design space allows the entity to be modeled to allow for the shadow and sunlight restrictions which may be imposed upon the site. The results of this interactive topological and geometrical editing is a solid model representation of the site constraints, and thus the allowable design space. Figure 142 above, illustrates the completed allowable design space. To begin the process in conceptual design within the allowable design space, figure 143 illustrates the introduction of a conceptual solid entity to the modeling environment.

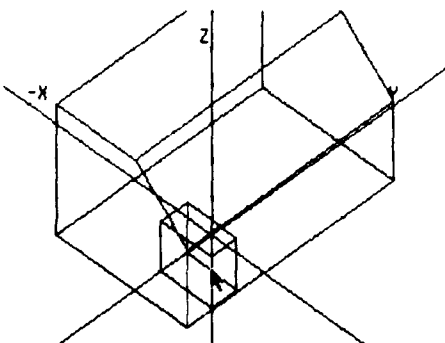


Figure 144: Translation of the solid entity within the design space, and restricted to the constraints of the design space.

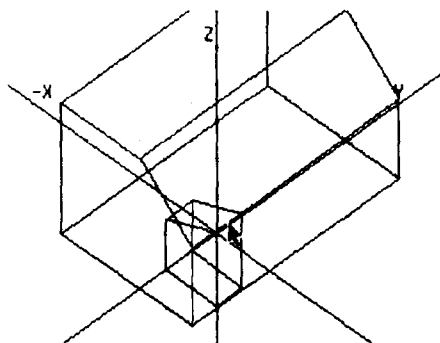


Figure 145: Interactive modeling of the solid entity within and constrained to the design space.

The interactive modeling process continues with the geometric editing of the entity within the allowable design space. Since the design space has a proximity zone established representing the side yard setback, geometric editing of the solid entity will not penetrate either this proximity zone or the established limits of the design space. Figure 144 illustrates the translation of the entity within, and constrained to, the allowable portions of the design space. Figure 145 illustrates the geometric editing of a segment which is constrained to the sunlight/shadow restrictions of the design space.

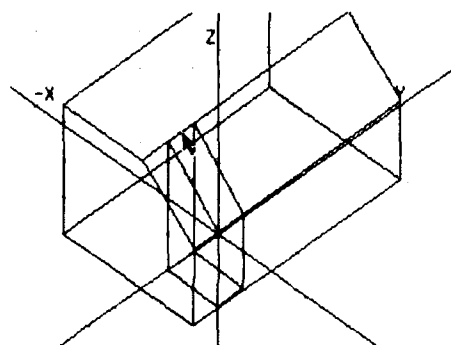


Figure 146: Subsequent geometric editing of the solid entity constrained to the height and shadow setback of the design space.

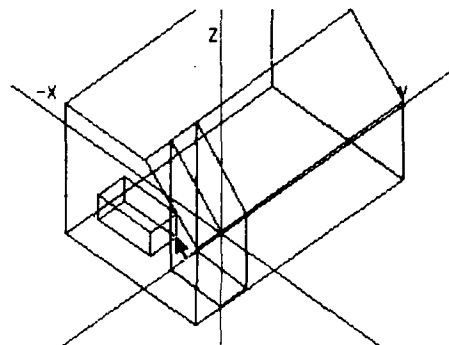


Figure 147: Creation of a second entity within the design space.

Additional geometric editing of the solid entity allows the form to be molded to the restrictions of the design space (figure 146). Since C*Mod constrains this modeling process interactively, the user can perform editing functions up to the bounds of any established constraint. The figures above illustrate this interactive geometric editing within the design space, and include the introduction of a second solid entity to the conceptual design process (figure 147).

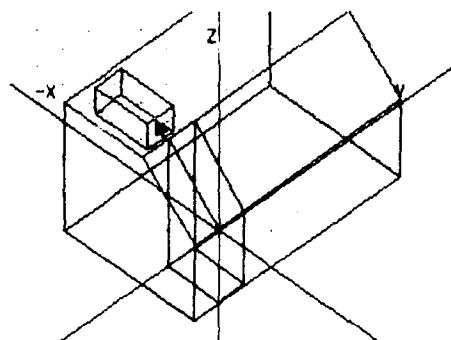


Figure 148: Vertical translation of the solid entity constrained to the height limitations of the design space.

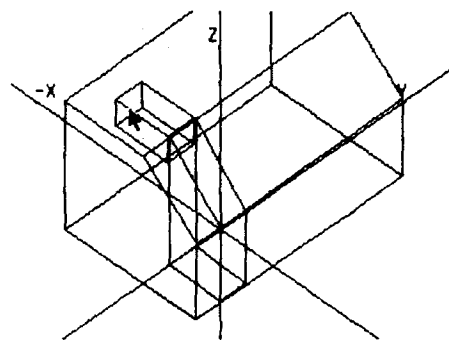


Figure 149: Association attribute of attraction established between the two solid entities.

Once the entity has been generated, relationship criteria can be established between the entities. Figures 148 and 149 above, illustrate the vertical translation of the entity to the height restriction of the design space, followed by the specification that the entity is to respond in an attraction association between it and the first entity created. By setting mobility of the first figure to fixed, any transformation of the second entity within an established tolerance from the first will cause it to snap up to the face and therefore connect the two entities. Providing this relationship enables the entities to act as a cohesive group, thus any transformation on either entity will effect both entities.

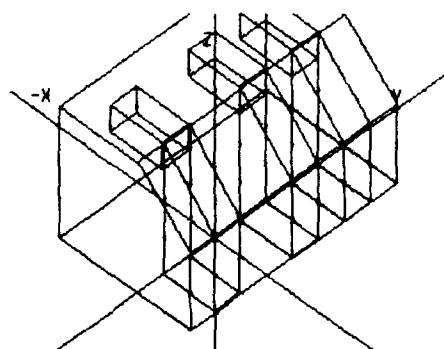


Figure 150: Generation of additional solid entities within the design space.

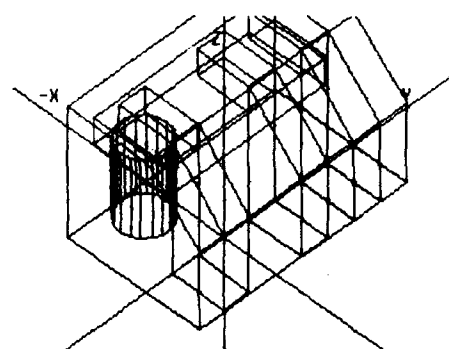


Figure 151: Completion of the conceptual design within the design space.

Continuing the process of generation and manipulation in a similar manner, additional conceptual entities can be introduced and modeled within the design space. Figures 150 and 151 illustrate the placement and editing of additional conceptual entities thus completing the conceptual design within an allowable design space. The completed conceptual design is illustrated in figure 152, and shows the solution with and without the design space visible. Since the modeler interactively restricts the generation and manipulation of entities within satisfaction of the established constraints, the conceptual design is known to be within the goals, objectives, and restrictions of the design space.

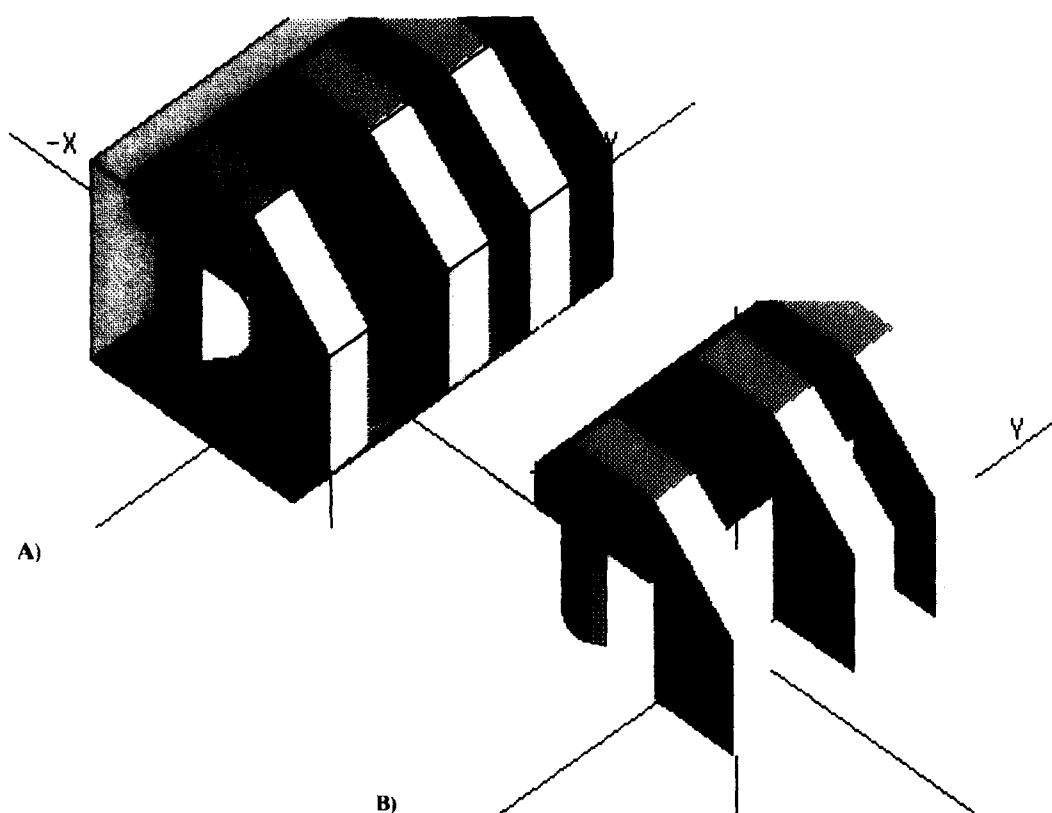


Figure 152: An illustration of the rendered results of the conceptual design created by the prototypical application C•Mod. A) Illustrates the conceptual design with the design space rendered. B) Illustrates the conceptual design without the design space.

5.4 Summary of Applications

The applications presented in this section are a small representation of the prototypical application C•Mod. C•Mod illustrates that the concepts of constraint-based solid modeling is a viable means of representing physical and relational design knowledge in an interactive solid modeling environment. The application demonstrates that the modeling of the physical interaction between solid entities, spaces, and constraints, can be applied to the early stages of the design process, including schematic design. The demonstration provides a means of exploration of the design envelope, or design space, in a means which is consistent with expected behavioral patterns of the entities generated and modeled. A designer is therefore free to create solid entities, or solids representing spatial entities, and model not only the specific geometry or topology, but the interaction between entities realistically. The results of which is that the computer is capable of providing the lower level decisions, such as encroachment upon a setback line, or the intersection of two solids, and adjusting, or reacting to the user in the form of an immediate response.

CHAPTER VI

EXTENSIONS AND FUTURE DIRECTIONS

During the course of its presentation, this thesis focused on a narrow spectrum of design knowledge which can be represented in the interactive three-dimensional modeling process. The scope of work was limited in nature due to time constraints. However, its purpose was to provide a functional prototype which could be utilized to illustrate the practicality of implementation. This does not suggest that this is all that can be done on the subject. On the contrary, there is much room for extension and expansion of thought both in terms of the prototypical application C•Mod, and the fundamental theory of constraint-based solid modeling. This chapter seeks to explore the frontiers of such direction by discussing several possible extensions to the prototypical application as well as extensions to the presented constraint-based modeling operations and theory. In addition, this chapter will discuss the anticipated future direction of constraint-based solid modeling and its future role in the interactive process of Computer-Aided Architectural Design.

6.1 Extensions to C•Mod

The prototypical application C•Mod is not the solution to the constraint-based question, there is room for improvement. This section elaborates on a two important areas which would enhance the application and provide a fully developed interactive modeler. The first, is extensions to the operations implemented, and includes the completion or implementation of the basic constraint operations discussed in Chapter II. The second, is to take advantage of

coherence in the implementation of the internal workings of the application. Both of these extensions are discussed in the following sections.

6.1.1 C•Mod Operations

Due to restricted time constraints, several basic operations were not implemented or fully developed. To provide a complete working prototype of the operations discussed in Chapter II, C•Mod would need to have these operations included. Among the operations left out are the spatial characteristics, which would support area and volumetric constraint modeling, rotational attributes, which would support three-dimensional rotational constraints, and containment relationships, which would allow the specification and ownership of elements. The implementation of these operations is not a major undertaking, since the user interface and internal representation are in place. However, implementation of these operations would greatly reduce the efficiency of the algorithms required to produce realistic real-time responses in the interactive modeling process.

6.1.2 Algorithm Coherence

C•Mod can effectively provide realistic responses to the interactions between entities in the current implementation. However, it is possible to enhance these operations by taking advantage of the coherence presented in the internal structure of the entities. One such coherence is provided in the method of determining when to transform secondary entities which intersect, or collide with the primary entity. Once interference is detected, transformation in the same direction implies that interference will always be detected. Therefore, taking advantage of this information will greatly reduce the number of calculations required during the transformation. Conversely, if an entity is transforming in a direction away from another entity, with the exception of the association relationships, there is no need to confirm that the entities do not interfere with one another, this is inherent as well. By taking

advantage of inherent nature of objects in three-space, the execution of the program could be greatly enhanced, and therefore provide a more realistic response in the interactive modeling process.

6.2 Extensions to Constraint-Based Modeling Operations

The extensions to the prototype application discussed above would allow the introduction of additional operations to the constraint-based modeling environment. This section discusses some of these extensions in general and includes the exploration of additional entity characteristic, relational characteristics, and physical laws of nature.

6.2.1 Entity Characteristic Operations

The entity characteristic operations presented as the basic set of constraints specifically apply to the physical description of the entity. These physical descriptions included representation, dimensional criteria, area and volumetric requirements, and mobility. This avenue of operations was selected as a fundamental set of operations which would illustrate the basic interactive nature of constraints upon the modeling process. In the event that the modeler was to assume a more realistic role, additional operations would be required. Operations such as mass, surface texture, rigidity, and selectable mobility. The following is a brief discussion of each of these possibilities.

The entity characteristic of mass would allow the modeler to introduce the natural laws of physics to the interactive process. The mass of an entity would determine potential energy which would be transferred during an interaction. Additionally, it would effect the momentum the entity gains and sustains during the interactive modeling process. The use of this characteristic would include modeling momentum of a structural member or component, and visualizing its reaction in a three-dimensional environment.

Surface texture would again allow the modeler to react to natural laws of physics. The surface qualities of an entity determine the friction generated when two objects are transformed along their faces, and would effect the relational characteristics of the modeling process. The use of this characteristic would include realistically modeling the interaction between elements of differing surface qualities such as a table leg being transformed along a carpeted floor.

Another physical characteristic which would benefit the constraint-based modeling approach is the use of rigidity. Rigidity would allow the modeling of materials with differing physical compositions. A wood column could be modeled with its base fixed in three-space, and interactively forced to bend until failure. The use of this characteristic is evident in the interactive modeling of structural components during the design process.

A final extension to the entity characteristic operations is that of specifiable mobility. Currently the mobility of an entity is fixed as an object. To realistically model the interaction with other entities, and utilizing some of the additional features listed above, the modeler must be able to fix any topological level of the entity in three-space. With this extension it would be possible to model the column example presented above by specifying that the bottom face of the column was to be fixed in three-space.

6.2.2 Relational Characteristic Operations

The interactive nature of a constraint-based modeler must allow, and provide for, relational characteristics between entities. This thesis proposed a few of the basic relationships which exist between entities such as containment, association, and proximity. In addition to these several other relationships could be modeled to enhance the realistic interactions between elements. Among those are ownership, grouping, and topological associations. Each of these additional operations are discussed below.

Ownership differs from containment in that an entity can own, or belong to another entity outside the bounds of a spatial representation. This relationship would establish a bond

between entities which would permit it to perform as a contiguous collection of entities while still retaining the individual characteristics. Thus a roof system composed of structural beams, joists, decking, and insulation could behave as a contiguous unit when being transformed in three-space while behaving in accordance with the requirements of each individual component.

Grouping would allow a collection of entities to be grouped together into a cohesive unit. This differs from ownership in that the individual characteristic are not retained. The collection of entities behaves as a single entity with its own set of constraints to be applied to the modeling process. In this relationship the components of a chair, arms, legs, seat, and back, could be grouped together and given its own set of constraints, or design knowledge, which identifies it as a chair, and therefore controls its behavior as a single entity.

The final additional relational operation is an extension to the existing association operation. By allowing the specification of associations between topological levels, an interactive modeler could associate, attract or repel, differing parts of the entity. Certain faces of an entity may be required to retain an attract association, such as the ends of a wall, while others may be required to retain a repel association, such as an opening for a door. By allowing this diversity within the entity itself, realistic situations can be modeled.

6.2.3 Physical Laws of Nature

To facilitate the realistic interaction between entities in an interactive modeler, the next logical extension would be to include the physical laws of nature to the modeling process. This is particularly important when determining the reactions from colliding entities. The resultant direction vector may not be in parallel with, and containing the same force of, a the moving entity. It is therefore important that when implementing any of the additional operations suggested, that the natural laws of physics be considered as a primary extension to the application.

6.3 Future Direction of Constraint-Based Solid Modeling

The potential for an interactive constraint-based solid modeling tool is clearly evident when considering the realistic interaction between entities, both in the physical and metaphysical sense. To bring solid modeling into the early stages of design, a modeler must be capable of supporting these interactive qualities to realistically model the environment. As the pretense for this thesis, interactive modeling within the guidelines established by the local zoning ordinances, building codes, and construction practices, provides a powerful foundation for future development of interactive solid modelers.

The future direction of constraint-based solid modeling stems from the ability to represent and model design knowledge to aid in the conceptual design process. This design knowledge, whether user specified, or read from a database containing the specific zoning ordinances, allows the modeler to perform as a powerful design tool in which the designer can call on and utilize that information in an interactive modeling platform. In this role, the constraint-based solid modeler provides an important tool which the designer can use to enhance the conceptual and schematic design process.

CHAPTER VII

CONCLUSIONS

The application of design knowledge to the interactive modeling process as illustrated and presented, provides a foundation for the exploration of realistically constraining the behavior of user definable three-dimensional geometric entities. The goal of this thesis, as stated in Chapter I, section 1.4, was to develop a constraint-based solid modeler for architectural applications which can be utilized as a tool for Computer-Aided Architectural Design. This thesis met that goal by successfully achieving the objectives stated. How the objectives were satisfied follows.

The first objective, was to provide the ability to generate, represent, and manipulate three-dimensional geometric entities through the use of a graphical interface. This included the facilities to store and support geometric and topological editing features at the point, segment, face, and volume levels. C•Mod provided this ability by supporting the internal solid data structure as specified in Chapter IV. The interactive nature of the generative and manipulative process of the modeler provided an excellent foundation for the implementation of constraint-based operations.

Secondly, the implementation was to allow for the specification of design knowledge applicable to user definable three-dimensional geometric entities such as solids or spatial representations. This included the ability to define, modify, store, and retrieve constraints upon the environment as well as the individual entities. Through the interactive user interface, C•Mod provided a dialog between the user and the application which facilitated this specification and manipulation of design knowledge. Additionally, the prototypical application

provided a means by which the entire set of constraints could be easily and permanently stored and retrieved at the discretion of the user.

The third objective was to provide the ability to manipulate the three-dimensional geometric entities in a manner which is consistent with the behavioral characteristics dictated by the entity specifications. As illustrated in Chapter III and Chapter IV, the prototypical application provided this realistic interaction of entities in an efficient and realistic manner. This interaction allowed the generation and manipulation of entities which facilitated the use of design knowledge to model a space constrained by realistic site requirements. The results of which allowed the interactive modeling within a design space.

Finally, the last objective was to provide the ability to extract information from an entity, which was provided by the specification of the entity. This was achieved by providing a query operation in the constraint mode. A user therefore has the ability to view the entire set of constraints, for any existing entities as well as the system specifications.

The primary expectation of this thesis was to contribute towards the development of an architectural solid modeler, which has the ability to represent information about a specific entity, as a foundation for design research, education, and practice. Four main goals were achieved in this research. One, support of the theoretical foundations which have preceded this research, including the support for the interactive constraint process of achieving problem satisfaction. Two, the delineation and definition of the basic components for a constraint-based solid modeler, including the representation of design knowledge, and the realistic interaction of conceptual entities. Three, the successful implementation and evaluation of a prototypical application, C*Mod, which illustrated concepts and theories presented. And four, the establishment of a strong foundation for further research and education in the use of interactive constraint-based systems in CAAD applications.

In 1973, Geoffrey Broadbent, discussed the use of computers to aid in the exploration and development of the design space interactively. By extending this early discussion to

include the generative and manipulative capabilities of a solid modeler, and modifying the theories of the modeling process to include interactive constraint satisfaction, design knowledge, and therefore constraints, can logically and efficiently enable a designer to bring the computer closer to the early stages of the design process. The ability to realistically represent the solid and spatial entities, as well as their behavioral patterns, provides an additional benefit of interacting with the modeler in a realistic manner. This thesis has illustrated that the application of design knowledge in C•Mod, although limited in scope, can provide an interactive modeling environment which allows design knowledge to constraint or limit the generation and manipulation of solid and spatial entities, and that the use of such a design tool has a strong and desirable link to the conceptual stages of the design process.

LIST OF REFERENCES

- AIA, (1987). "Building design." In *American Institute of Architects - Handbook of Professional Practice*. 11th ed., Ch 2.5, American Institute of Architects.
- Addison, M.S. (1988). *A Multiple Criteria Satisficing Methodology for the Design of Energy-Efficient Buildings*, Master's Thesis, Arizona State University.
- Adeli, H. (1990). *Knowledge Engineering, Volume I, Fundamentals*. McGraw-Hill, New York.
- Adeli, H. (1990). *Knowledge Engineering, Volume II, Applications*. McGraw-Hill, New York.
- Akin, O. (1986). *Psychology of Architectural Design*, Pion Limited, Brondesbury Park, London.
- Angel, E. (1990). *Computer Graphics*, Addison-Wesley Publishing Company, Reading, Massachusetts.
- Archer, L.B. (1969). "The structure of the design process." In G. Broadbent and A. Ward (eds.), *Design Methods in Architecture*, Lund Humphries, London, pp. 76-102.
- Archer, L.B. (1970). "An overview of the structure of the design process." In G.T. Moore (ed.), *Emerging Methods in Environmental Design and Planning*, The MIT Press, Cambridge, Massachusetts, pp. 285-307.
- Asimow, W. (1964). *Introduction to Design*, Prentice-Hall, Englewood Cliffs, New Jersey.
- Berger, M. (1986). *Computer Graphics With Pascal*, The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California.
- Bijl, A. (1987). "Strategies for CAD." In *Proceedings from the First Eurographics Workshop on Intelligent CAD Systems*, Noordwijkerhout, The Netherlands, (April)
- Broadbent, G. (1973). *Design in Architecture*, John Wiley & Sons, London.
- Carrara, G., and G. Novembri (1985). "Constraint-bounded design search." In A. Pipes (ed.), *Computer-Aided Architectural Design Futures*, Butterworths, London, pp. 146-157.
- Chernicoff, S. (1987). *Macintosh™ Revealed, Volume One: Unlocking the Toolbox*, Haden Books, Indianapolis, Indiana.

- Chernicoff, S. (1987). *Macintosh™ Revealed, Volume Two: Programming with the Toolbox*, Haden Books, Indianapolis, Indiana.
- Cross, A. (1986). "Design intelligence: the use of codes and language systems in design." In *Design Studies*, vol 7, no 1, January.
- Cohon, J.L. (1978). *Multiobjective Programming and Planning*, Academic Press, New York.
- Coyne, R.D. and J.S.Gero, (1985). "Design knowledge and sequential plans." In *Environment and Planning B*, 12:401-418.
- Coyne, R.D. and J.S.Gero, (1986). "Semantics and the Organization of knowledge in design." In *Design Computing*, 1(1):68-89.
- Coyne, R.D., M.A.Rosenman, A.D.Radford, M.Balachandran, and J.S.Gero. (1990). *Knowledge-Based Design Systems*, Addison-Wesley, Reading, Massachusetts.
- Debenham, J.K. (1989). *Knowledge Systems Design*. Richard P. Brent (ed.), Prentice Hall, New York.
- Eastman, C. M. (1970). "On the analysis of intuitive design processes." In G.T. Moore (ed.), *Emerging Methods in Environmental Design and Planning*, The MIT Press, Cambridge, Massachusetts, pp. 38-47.
- Eastman, C.M. (1985). "Abstractions: a conceptual approach for structuring interaction with integrated CAD systems." In *Computers and Graphics*, 9(2):97-105.
- Eastman, C.M. (1986). "Fundamental problems in the development of computer-based architectural design models." In *Proceedings, The Computability of Design*. 1986 SUNY Buffalo Symposium on Computer-Aided Design.
- Foley, J. D., A. van Dam, S.K.Fiener, J.F.Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company, Reading Massachusetts.
- Gero, J.S. (1985). "An overview of knowledge engineering and its relevance to CAAD." In A.Pipes (ed.), *Computer-Aided Architectural Design Futures*, Butterworths, London, pp. 107-119.
- Gips, J., and G.Stiny. (1980). "Production systems and grammars: a uniform characterization." In *Environment and Planning B*, 7:399-408.
- Glassner, A.S. (1989). *3D Computer Graphics, A User's Guide for Artists and Designers*, Design Press, New York.
- Gross, M., S.Ervin, J.Anderson, A.Fleisher, (1986). "Designing with constraints." In *Proceedings, The Computability of Design*. 1986 SUNY Buffalo Symposium on Computer-Aided Design.
- Kalay, Y.E. (1985). "Redefining the role of computers in architecture: from drafting/modeling tools to knowledge based design assistants." In *Computer Aided Design*, vol.17, no.7. September.

- Kalay, Y.E. (1989). *Modeling Objects and Environments*, John Wiley & Sons, New York.
- Landsdown, J. (1985). "Requirements for knowledge-based systems in design." In A. Pipes (ed.), *Computer-Aided Architectural Design Futures*, Butterworths, London, pp. 120-127.
- Leon, S. J. (1986). *Linear Algebra with Applications*, Macmillan Publishing Company, New York, 2nd edition.
- Maher, M.L. (1984). *HI-RISE: An Expert System for the Preliminary Structural Design of High Rise Buildings*, Ph.D. Thesis, Department of Civil Engineering, Carnegie Mellon University, Pittsburgh.
- Maher, M.L. (1985). "HI-RISE and beyond: directions for expert systems in design." In *Computer Aided Design*, vol 17, no 9, November.
- March, L., and G.Stiny, (1985). "Spatial systems in architecture and design: some history and logic." In *Environment and planning B*, 12:31-53.
- McIntosh, P.G. (1986). "Models of spatial information in computer-aided architectural design: a comparative study." In *Proceedings, The Computability of Design*. 1986 SUNY Buffalo Symposium on Computer-Aided Design.
- Mednieks, Z.R., and T.M. Mednieks (1989). *C Programming Techniques for the Macintosh®*, Howard W. Sams and Company, Carmel, Indiana.
- Mitchell, W.J., J.P.Steadman, and R.S.Liggett. (1976). "Synthesis and optimization of small rectangular floor plans." In *Environment and Planning B*. 3:37-70.
- Mitchell, W.J. (1977). *Computer-Aided Architectural Design*, Petrocelli/Charter, New York.
- Mitchell, W.J. (1986). "Reasoning about form and function." In *Proceedings, The Computability of Design*. 1986 SUNY Buffalo Symposium on Computer-Aided Design.
- Mitchell, W.J. (1990). *The Logic of Architecture*, The MIT Press, Cambridge, Massachusetts.
- Mortenson, M. E. (1985). *Geometric Modeling*, John Wiley and Sons, New York.
- Negroponte, N. (1972). *The Architecture Machine*, The MIT Press, Cambridge, Massachusetts.
- Negroponte, N. (1975). *Soft Architecture Machines*, The MIT Press, Cambridge, Massachusetts.
- Pang, G.K.H., and A.G.J. MacFarlane, (1987) "An Expert Systems Approach to Computer-Aided Design of Multivariable Systems." In M.Thoma and A. Wyner (eds.) *Lecture Notes in Control and Information Sciences*, Springer-Verlag, Berlin.
- Payne, E.C., and R.C. McArthur, (1990). *Developing Expert Systems, A Knowledge Engineer's Handbook for Rules and Objects*, John Wiley & Sons, Inc, New York.

- Plastock, R.A. and G.Kalley, (1986). *Computer Graphics*, Schaum's outline series in computers, McGraw-Hill Book Company, New York.
- Pugh, K. (1990). *All on C*, Scott, Foresman/Little, Brown Higher Education, Glenview, Illinois.
- Radford, A.D., and J.S.Gero, (1985). "Towards generative expert systems for architectural detailing." In *Computer Aided Design*, vol 17, number 9, November.
- Radford, A.D., and J.S.Gero, (1988). *Design by Optimization in Architecture, Building and Construction*, Van Nostrand Reinhold, New York.
- Reitman, W.R. (1964). "Heuristic decision procedures, open constraints, and the structure of ill-defined problems." In M.W.Shelley, and G.L.Bryan, (eds.), *Human Judgements and Optimality*, Wiley, New York, pp. 282-315.
- Rich, E. (1983), *Artificial Intelligence*. McGraw Hill, New York.
- Rittel, H.W.J. (1971). "Some principles for the design of an educational system for design." In *Journal of Architectural Education*, vol 26, no 1, pp. 16-26.
- Rogers, D. F. (1985). *Procedural Elements for Computer Graphics*, McGraw-Hill Publishing Company, New York.
- Rosenman, M.A., and J.S.Gero, (1985). "Design codes as expert systems." In *Computer Aided Design*, 17(9):399-409.
- Schildt, H. (1987). *Artificial Intelligence using C, The C Programmer's Guide to Artificial Intelligence Techniques*. Osborne McGraw-Hill, Berkeley, California.
- Shapiro, S.C., and J.Geller, (1986). "Artificial intelligence and automated design." In *Proceedings, The Computability of Design*. 1986 SUNY Buffalo Symposium on Computer-Aided Design.
- Simon, H.A. (1983). "Search and reasoning in problem solving." In *Artificial Intelligence*, vol 21, pp. 7-29.
- Spiegel, M.R. (1968). *Mathematical Handbook of Formulas and Tables*, Schaum's outline series in mathematics, McGraw-Hill Book Company, New York.
- Sussman, G.J. and G.Steele, (1980). "CONSTRAINTS - A language for expressing almost hierarchical descriptions." In *Artificial Intelligence* August 1980, 1-40.
- Swerdloff, L.M, and Y.E.Kalay, (1986). "A partnership approach to computer-aided design." In *Proceedings, The Computability of Design*. 1986 SUNY Buffalo Symposium on Computer-Aided Design.
- Tomiyaama, T. and P.J.W. ten Hagen, (1987). "The Concept of Intelligent Integrated Interactive CAD Systems, CWI Report No. CS-R8717." In *Center for Mathematics and Computer Science*, Amsterdam, (April).

- Tomiyama, T. and P.J.W. ten Hagen, (1987). "Organization of Design Knowledge in an Intelligent CAD Environment, CWI Report No. CS-R8720." In *Center for Mathematics and Computer Science*, Amsterdam, (April).
- Veklerov, E. and O.Pekelny, (1989). *Computer Language C*, Harcourt Brace Javanovich College Outline Series, Harcourt Brace Javanovich, Publishers, New York.
- Walters, J. and N.R.Nielson, (1988). *Crafting Knowledge Based Systems, Expert Systems Made Realistic*. John Wiley & Sons, New York.
- Watt, A. (1989). *Fundamentals of Three-Dimensional Computer Graphics*, Addison Wesley Publishing Company, Wokingham, England.
- Woodwark, J. (1989). *Geometric Reasoning*, Oxford University Press, New York.
- Yessios, C.I. (1983). "Architectural modeling: eliminating the design/drafting split in CAAD." In *National Computer Graphics Association Conference*, June.
- Yessios, C.I. (1987). "Architectural modeling and knowledge systems." In *Proceedings, NCGA Computer Graphics*.